משה יצחקי

שפת C

נושאים מתקדמים ומולטימדיה

גרפיקה ושילוב תמונות

השמעת קבצי קול ותקליטורים

אנימציה והנפשה

תקשורת בין מחשבים

ונושאים רבים נוספים



הוצאת הוד-עמי
לספרי מחשבים



שפת C
נושאים מתקדמים ומולטימדיה

קרא על התקליטור בהקדמה, בנספח ו' ו-ז'
ובקובץ ONCD שבתקליטור

עורך ראשי: **יצחק עמיהוד**

עריכה ועיצוב: **שרה עמיהוד, טליה טופז**

עיצוב עטיפה: **שרון רז**

לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.

- **טלפון: 09-9564716**
- **פקס: 09-9571582**
- **דואר אלקטרוני: info@hod-ami.co.il**
- **אתר באינטרנט: www.hod-ami.co.il**

שפת C

נושאים מתקדמים
ומולטימדיה

משה יצחקי



C Language - Advanced topics & Multimedia
By M. Itzhaki

הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 6108 הרצליה 46160
טלפון: 09-9564716 פקס: 09-9571582

www.hod-ami.co.il
info@hod-ami.co.il

הודפס בישראל 2000

HOD-AMI Ltd.
P.O.B. 6108, Herzliya
ISRAEL, 2000

ISBN 965-361-251-4 מסת"ב

תוכן עניינים מקוצר

תוכן עניינים

פרק 15: השמעת קבצי WAV 288

פרק 16: הקלטת קבצי WAV 329

פרק 17: תקשורת טורית בין מחשבים והפעלת המודם 339

הקדמה

הספר "**שפת C - נושאים מתקדמים ומולטימדיה**" מיועד לתלמידי תיכון, מכללה ואוניברסיטה, המגישים עבודות גמר בשפת C, ולכל מי שרוצה להעשיר את הידע שלו במיגוון נושאים מתקדמים בתכנות שעד כה לא הופיעו בספרי לימוד אחרים.

ספר זה **אינו** ספר לימוד של שפת C. קהל היעד העיקרי שלו הם המתקדמים, בעלי ידע בסיסי בתכנות. עם זאת, הפרק הראשון בספר מהווה תקציר וחזרה על שפת C, וכך הוא מאפשר גם למתכנתים בעלי ידע בשפות אחרות להכיר את שפת C וללמוד בספר זה. גם אלה שמכירים את שפת C ורוצים לרענן את ידיעותיהם בשפה יוכלו למצוא בפרק זה חומר לימודי שיענה על דרישותיהם. מי שאינו בקיא בשפת תכנות כלשהי, מומלץ שיפנה תחילה לאחד מהספרים האלה בהוצאת הוד-עמי:

- **המדריך השלם לשפת C** / משה ליכטמן, עמית רש.
- **ללמוד C** / יואב נתיב.

חמשת הפרקים הראשונים בספר מציגים נושאים חשובים בשפת C שאינם מקובלים בדרך כלל בספרי לימוד של השפה (כגון עבודה בסיביות, שימוש בפסיקות, פלישה לפסיקות, יצירת פרויקט ועוד). גם מי שמכיר נושאים אלה, חשוב שיקרא אותם, כיון שהם מהווים את הבסיס ללימוד ותרגול הנושאים המתקדמים הנוספים שבספר.

שאר הפרקים מציגים מיגוון נושאים מתקדמים בתכנות ובמולטימדיה. הנושאים ממומשים אמנם בשפת C, אך החומר התיאורטי מתאים לשפות תכנות אחרות גם כן, **ולכן ניתן יהיה לשכתב את קטעי הקוד בקלות ולהתאימם גם לשפות תכנות אחרות.**

בסוף כל פרק ישנה **יחידת תכנות** (Unit) המסכמת את כל החומר הנלמד בפרק ומכילה גם פתרונות לחלק מהמשימות שפורטו בו. לאחרי כל יחידה יש **תוכנית דוגמה** המשתמשת ביחידה זו ומדגימה את השימוש בה.

היחידות והתוכניות שבכל פרק, נמצאות גם **בתקליטור המצורף לספר זה** (תחת התיקיה Books\59281). לכל פרק בספר יש תיקיה נפרדת המכילה את כל קבצי הקוד הרלוונטיים לו. קטעי קוד אלה נכתבו בספר עם הצללה (רקע אפור). התוכניות שבתקליטור נכתבו כפי שהן צריכות להיראות בעבודת גמר בעת הלימוד וגם בעבודה מעשית. כך תוכלו להתרגל למראה הכללי של הקוד (מסגרות עזר, הזחות) ולתיעוד. בספר, בפרקים 3 ו- 4, הרחבתי כיצד צריכה להיראות תוכנית ואח"כ עברתי למבנה שבדרך כלל נהוג בספרות המקצועית.

פנה לנספח ו' כדי לקרוא כיצד להריץ את תוכניות הדוגמה שבתקליטור, בעזרת המהדר Borland C/C++ 3.x.

היחידות שבסוף כל פרק מאפשרות גם למי שאינו רוצה להתעמק בדרך המימוש של הפונקציות השונות, ללמוד כיצד לשלב אותן ולהשתמש בהן לפתרון הבעיה המוצגת. עם זאת, בעת הגשת עבודה במסגרת לימודים, ובמיוחד בעת פיתוח תוכנה, רצוי ואף חובה להכיר ולהבין את תוכנם של הפונקציות.

למה כדאי ללמוד את הנושאים האלה?

רבים מביניכם בוודאי תוהים, מדוע צריך ללמוד נושאי מולטימדיה בשפת C. הרי כיום ניתן לשלב תמונות וקבצי קול בפרויקט בקלות רבה בשפות ויזואליות, כמו Visual C++ ו-Visual Basic.

גם היום נכתבות מערכות תוכנה רבות ומקיפות בשפת C. כך לדוגמה, תוכנות Windows כתובות בשפת C, תוכנות Linux ואחרות. גם כשכותבים תוכנית בשפה ויזואלית ועוזרים בפקדים שבסרגל הכלים בכדי לשלב בתוכנית קבצי מולטימדיה, ראוי לדעת שהפקדים נכתבו במקור בשפת C. יתרה מכך, כשרוצים לכתוב פרויקט נרחב אשר הביצועים ומהירות התגובה בו חשובים, כותבים את הפרויקט בשפת C. מסיבות אלו, שפת C עדיין רחוקה מלהיעלם מעולם התכנות, ומשמשת שפת בסיס ללימוד תכנות בבתי ספר מקצועיים, במכללות ובאוניברסיטאות.

פרקי המולטימדיה בספר זה מכילים תמצית של החומר התיאורטי הרב הקשור לנושאים אלה. כדי ללמוד את כל החומר הקשור להם דרושים 17 ספרים ולא 17 פרקים בלבד. לדוגמה, בפרק 9 העוסק בגרפיקה תכירו מוד של גרפיקה ב-256 צבעים. עם זאת, ידוע שקיימים מודים רבים אחרים התומכים ברזולוציה גבוהה ובמספר צבעים רב יותר. בחרתי להציג את האפשרות הבסיסית מכיון שהיא פשוטה לתכנות, והדגמתי בעזרתה נושאים כלליים בגרפיקה הנכונים גם עבור מודים אחרים (כגון דפים וירטואליים והשהיית תותח האלקטרונים). באופן זה, תוכלו לקבל את הרקע והבסיס לנושא ואם תרצו – תוכלו לכתוב בעצמכם יחידה התומכת במוד יותר מתקדם.

באותה מידה, בפרק העוסק בהצגת תמונות הדגמתי את הנושא בעזרת תמונות BMP; ובפרק העוסק בהשמעת קבצי קול הדגמתי את הנושא בעזרת פורמט בסיסי של קבצי WAV. בכל פרק הדגמתי את הדרוש בעזרת הדוגמה הקלה ביותר למימוש והסברתי כיצד ניתן לשדרג את הקוד כדי שיתמוך במבני תכנות מתקדמים יותר.

הפרקים הם עצמאיים בדרך כלל, והכוונה לכך שאין חובה לקרוא אותם לפי סדר מחייב כלשהו. עם זאת, רצוי לקרוא את הפרק כולו מתחילתו לסופו, כי **המטרה המסתתרת מאחורי נושאים אלו היא להדגים כיצד להתמודד עם בעיות שונות בתכנות, וכיצד לגשת לכתיבת קוד מורכב ולעיתים גם מסובך**. **חשוב שיהיה לכל מתכנת ידע בסיסי בכל אחד מהנושאים הללו, גם אם אינו מתכוון לעסוק בהם בעתיד**. כך לדוגמה, גם מי שכותב בשפת תכנות ויזואלית ומשתמש בפקד מוכן להשמעת קובץ WAV, יבין כיצד הקובץ מושמע (בעזרת מנגנון פסיקות עצמאי) ומדוע גם אם התוכנית נתקעת, המנגינה עדיין נשמעת.

הנושאים בספר כתובים על פי סדר קושי עולה, ולכן גם מי שאינו מעוניין בכל הנושאים, רצוי שיקרא את הפרקים המעניינים אותו, על פי הסדר בו הם כתובים בספר. בכל מקרה, רצוי להתחיל מחמשת הפרקים הראשונים המהווים את הבסיס לכל הפרקים שבהמשך.

רעיונות לפרויקטים שונים שניתן להכין בעזרת ספר זה

- תוכנה לטיפול בקבצים וספריות בסגנון Norton Commander, המאפשרת לסייר בכוננים, ולהפעיל, להעתיק ולמחוק קבצים ותיקיות.
- ארגונית – יומן אלקטרוני הכולל: לוח פגישות יומי, התרעות על ימי הולדת ואירועים חשובים, ורשימת מטלות יומית.
- ספר טלפונים ממוחשב המשלב חייגן טלפון המאפשר לחייג ישירות מתוך ספר הטלפונים הממוחשב בעזרת מודם.
- מעבד תמלילים בסיסי המאפשר העתקה, מחיקה, שמירה, טעינה, הדפסה ופעולות Undo.
- משחק כגון 'שולה מוקשים', או 'סוגר שטחים' החושף תמונת BMP הנמצאת ברקע.
- תוכנה להשמעה והקלטה של קבצי קול.
- תפריט קיצורים המיועד למערכת רבת משתמשים, המציג לכל משתמש את תפריטי הקיצורים לתוכניות בהם הוא משתמש.
- נגן תקליטורים המאפשר השמעת תקליטורים, הכנת רשימת השמעה והשמעה אקראית של שירים.
- סיפור מולטימדיה לילדים. תוכן הסיפור מוצג על המסך וברקע נשמע הקריין המקריא את תוכן הסיפור.
- תוכנת ציור (בסגנון הצייר של windows) המאפשרת לצייר צורות גיאומטריות שונות, להעתיק ולמחוק קטעים מהתמונה, ולשמור את התמונה כתמונת BMP.
- מהדר בסיסי בעברית, המאפשר כתיבת תוכניות בעברית מבנית (במקום בשפת תכנות).
- תוכנה לניהול ספריית וידאו, המנהלת מעקב אחר מלאי הסרטים בספריה ומאפשרת לבצע השאלות והחזרות של סרטים ומפיקה דוחות שונים בהתאם.
- תוכנת תקשורת בין מחשבים בסגנון (PC anywhere) הכוללת צ'אט בין מחשבים המחוברים בעזרת כבל או בעזרת מודם, או לחילופין תוכנה להעברת קבצים.
- לומדה כלשהי המיועדת לילדים ללימוד חשבון, צורות, צבעים, קריאה וכדומה. ניתן לשלב אף קבצי קול בהם תקליטו את עצמכם נותנים הוראות לילד המשתמש בלומדה.

ולסיום

הספר מוקדש **לכם** הקוראים, בתקווה שיוכל להקל עליכם את כתיבת הפרויקטים בעת הלימודים ובעבודה מעשית, ויעשיר אתכם בידע חשוב במיגוון רחב של נושאי תכנות.

תודה לכל מי שתמך וייעץ לי במהלך כתיבת הספר, ובמיוחד ל**הוריי** ולמר **חיים ויינגולד.**

אם יש לכם רעיונות, הערות, הצעות ושאלות תוכלו לפנות אלי בדואר אלקטרוני, לכתובת **AdvancedC_Book@hotmail.com**.

אשתדל לענות לכל אחד שיפנה, ובמידת הצורך, לשלב את ההערות במהדורות הבאות של הספר.

שלכם,

משה יצחקי
תמוז, תש"ס
7.2000

פרק 1

תקציר פקודות בשפת C

פרק זה הוא **חזרה** על אבני הבניין של **שפת C**, וכולל תקציר פקודות בסיסיות. הפרק מיועד לבעלי ידע מוקדם בשפת C או בשפות תכנות אחרות.

להסברים מעמיקים על רזי השפה, ניתן לפנות לספרים האלה:

- **המדריך השלם לשפת C** / משה ליכטמן, עמית רש
- **ללמוד C** / יואב נתיב
- **שפת C - תוכניות ופתרונות** / אלון מנור

הספרים ב**הוצאת הוד-עמי** לספרי מחשבים.

ההסברים על הפקודות מלווים בדוגמאות רבות, אשר מסודרות בטבלה בעלת שני טורים, הטור הימני מתאר את הפקודה והטור השמאלי מציג את דרך כתיבתה.

1.1 הערות כלליות

- כל פקודה צריכה להסתיים בנקודה-פסיק (;).
- בכל מקום בו כתוב "פקודה" ניתן לכתוב גם בלוק (גוש) של פקודות, ולא פקודה אחת בלבד.
- בלוק פקודות הינו פקודה אחת או יותר, אשר תחומות בין זוג סוגריים מסולסלים ({ }).
- הערה כותבים אחרי שני קווים נטויים (//) או שתוחמים אותה כך: /* הערה */.
- בשפת C יש הבחנה בין אותיות גדולות וקטנות, ולכן יש להקפיד על כך בעת כתיבת משתנים.
- בשפת C אין משתנים בוליאניים. כל משתנה שערכו שונה מאפס יוכר כ"אמת", ומשתנה שערכו אפס יוכר כ"שקר".

1.2 פקודות בסיסיות

הכרזות והצהרות

סדר כתיבת הפקודות בתוכנית הוא כסדר הפקודות בטבלה.

הכרזה על יחידה (unit) מהספריה הסטנדרטית.	**#include** <שם_היחידה.h>
הכרזה על יחידה שנבנתה על ידי המתכנת.	**#include** "שם_היחידה.h"
הגדרת קבועים.	**#define** שם ערך
הצהרת משתנים* (ראו סוגי משתנים בתרשים 1.1).	סוג שם, שם, שם... ;

* בהצהרת המשתנים ניתן גם לאתחל אותם בערך ראשוני.

סוג משתנה	גודל בבתים	תחום
char	1	-128 → 127
unsigned char	1	0 → 255
short	2	-32,768 → 32,767
int	2	-32,768 → 32,767
unsigned int	2	0 → 65,535
long	4	-2,147,483,648 → 2,147,483,647
unsigned long	4	0 → 4,294,967,295
float	4	$3.4x10^{-38} \rightarrow 3.4x10^{38}$
double	8	$1.7x10^{-308} \rightarrow 1.7x10^{308}$

תרשים 1.1
הגדלים שבתרשים נכונים ל-MS-DOS

הצבות

הצבה רגילה מתבצעת באמצעות סימן השוויון (=) (ולא על ידי נקודתיים+שווה (:=) כמו בפסקל).

ראו רשימת אופרטורים אריתמטיים בתרשים 1.2.

אופרטור	משמעות
+	חיבור
-	חיסור
*	כפל
/	חילוק
%	שארית

תרשים 1.2

בשפת C ניתן לבצע **הצבה מקוצרת** על ידי כתיבת האופרטור לפני סימן השוויון.

במקום לכתוב: `x = x +10 ;`

נוכל לכתוב: `x += 10 ;`

ומכאן, **הגדלה/הקטנה עצמית** ב-1 אפשר לכתוב בדרך זו:

x**++** שקול ל- **++**x שקול ל- x**+=**1 שקול ל- x=x+1

x-- שקול ל- --x שקול ל- x-=1 שקול ל- x=x-1

ההבדל בין כתיבת צמד האופרטורים לפני או אחרי המשתנה מתבטא בביטויים מורכבים. אם האופרטורים נכתבים "אחרי", הביטוי יחושב על פי הערך המקורי של המשתנה ורק אחר כך ישתנה ערך המשתנה. אם האופרטורים נכתבים "לפני", הביטוי יחושב על פי הערך החדש של המשתנה.

לעיתים נרצה להמיר ערך של משתנה מסוג אחד למשתנה מסוג אחר. לשם כך נשתמש ב-**הסבה בכוח** (**Casting**). נעשה זאת כך:

```
משתנה-מסוג-א = (סוג-א) משתנה-מסוג-ב
```

לדוגמה:

```
int i; float f;

i = (int)f ;
```

פקודות קלט/פלט

פקודות קלט/פלט המפורטות בסעיף זה דורשות הכרזה על יחידות הספריה <stdio.h> ו- <conio.h>.

פעולת קלט.	`scanf("מחרוזת בקרה", &משתנה, ... &משתנה);`
פעולת פלט.	`printf("מחרוזת בקרה", משתנה, ... משתנה);`

מחרוזות הבקרה מורכבת מהמלל שאנו רוצים שיוצג על המסך, תווי בקרה עבור כל משתנה שאת ערכו אנו רוצים להציג (ראה תרשים 1.3) וקבועים מיוחדים (תרשים 1.4). אחרי מחרוזות הבקרה נרשום את רשימת המשתנים/ביטויים, שאת ערכם אנו רוצים להציג (בהתאם לסדר תווי הבקרה שרשמנו). המשתנים יופרדו בפסיקים. לדוגמה:

```
scanf("%d %d",&a,&b);
printf("The sum of %d and %d is %d\n",a,b,a+b);
```

בהנחה שקלטנו ל-a את הערך 3 ול-b את הערך 2, יוצג:

```
The sum of 3 and 2 is 5
```

תו בקרה	משמעות
%c	ייצוג תו אסקי בודד.
%d	ייצוג עשרוני של שלם רגיל.
%e	ייצוג עשרוני במבנה אקספוננט (exponent). האות e תראה קטנה.
%E	ייצוג עשרוני במבנה אקספוננט (exponent). האות E תראה גדולה.
%f	ייצוג שבר עשרוני מסוג float.
%ld	ייצוג עשרוני של שלם ארוך.
%lf	ייצוג שבר עשרוני מסוג double.
%lu	ייצוג עשרוני של שלם ארוך חיובי (בלי סימן).
%o	ייצוג אוקטלי (בסיס 8).
%p	ייצוג של מצביע.
%s	ייצוג של מחרוזת תווים.
%u	ייצוג עשרוני של שלם רגיל חיובי (לא מסומן).
%x	ייצוג הקסאדצימלי (בסיס 16). האותיות a-f יראו קטנות.
%X	ייצוג הקסאדצימלי (בסיס 16). האותיות A-F יראו גדולות.

תרשים 1.3

קבוע	משמעות
\n	עבור לתחילת השורה הבאה.
\r	חזור לתחילת השורה הנוכחית.
\t	עבור לטאב הבא.
\b	חזור תו אחד אחורה.
\\	הדפסת קו נטוי (\).
\'	הדפסת גרש (').

תרשים 1.4

קליטת תו בודד ללא הצגתו על המסך.	משתנה = **getch();**
קליטת תו בודד והצגתו על המסך.	משתנה = **getche();**
הצגת תו בודד על המסך.	**putch(**משתנה**);**

תנאים

התניה. התנאי שבסוגריים מורכב משני משתנים, אשר ביניהם יש אופרטור לוגי (תרשים 1.5), או לחילופין ממשתנה אחד (אם ערכו שונה מאפס, התנאי מתקיים).	**if** (תנאי) פקודה; **else** פקודה;

ניתן לכתוב תנאי מורכב, שבו יש רצף של תנאים וביניהם האופרטור "וגם", או האופרטור "או" (תרשים 1.5). כדי לקבוע את סדר בדיקת התנאים ניתן להשתמש בסוגריים, כמקובל באריתמטיקה.

אופרטור	משמעות
<	קטן מ-
>	גדול מ-
==	שווה ל-
<=	קטן שווה ל-
>=	גדול שווה ל-
!=	לא שווה ל-
!	לא (NOT)
&&	וגם (AND)
||	או (OR)

תרשים 1.5

דוגמה:

```
if ( x && x<=3 && x>-3 )
  printf ("תנאי זה יתקיים עבור הערכים:-2,-1,1,2,3");
```

רב-ברירה. ברב-ברירה אנו בודקים את כל הערכים האפשריים שיכול להכיל משתנה מסוים. אם המשתנה מכיל את הערך הכתוב אחרי הפקודה case, יתבצעו כל הפקודות הכתובות אחריו, עד הפקודה break אשר מסמלת את סיום האפשרות. ללא פקודה זו יתבצעו גם כל הפקודות שבהמשך. ניתן לנצל עובדה זו כדי לשרשר קטעים עוקבים. אם ערך המשתנה אינו תואם אפשרות כלשהי, יתבצע קטע התוכנית שאחרי הפקודה default.	**switch** (משתנה) **{** **case** ערך : פקודה; פקודה;פקודה;... **break**; **case** ערך : פקודה; פקודה;פקודה;... **break**; . . . **default** : פקודה; פקודה;פקודה;... **}**

לולאות

הלולאה while כל עוד התנאי מתקיים, הפקודה תתבצע.	`while (תנאי)` `    פקודה;`
הלולאה do-while הפקודה תתבצע כל עוד התנאי מתקיים.	`do` `    פקודה;` `while (תנאי);`
הלולאה for הלולאה מתבצעת בסדר זה: אתחול, בדיקת תנאי, ביצוע פקודה, קידום, בדיקה, ביצוע, קידום... (עד שהתנאי יפסיק להתקיים).	`for (אתחול;תנאי;קידום)` `    פקודה;`

דוגמאות להדפסת הערכים בין 1 ל-10 (כולל):

```
i = 1;
while (i<=10)
  printf("%d",i++);

for (i=1;i<=10;i++)
  printf("%d",i);
```

מערכים

מערך חד-מימדי. N מייצג את מספר האיברים במערך. אינדקס האיבר הראשון הוא אפס (0) ואינדקס האיבר האחרון הוא N-1. פנייה לאיבר במערך היא באמצעות ציון שם המערך ואחר כך מספר האינדקס בסוגריים מרובעים.	`סוג שם [N] = {ערך,ערך...} ;`
מערכים רב-מימדיים שימו לב, גם כאן טווחי האינדקסים בין 0 לבין "N-1". "ערך" מייצג את N, מספר האיברים במערך.	`סוג שם [ערך][ערך]... =` `  { {ערך,ערך...},` `    {ערך,ערך...},` `    {ערך,ערך...}...} ;`

דוגמה:

```
int Mat [4][2] =
     { {1,2},{3,4},{5,6},{7,8} };

printf ("%d",Mat[2,1]);              //6 יודפס
```

מחרוזות

`char שם [N] = "מחרוזת" ;`	**מחרוזות**. מחרוזות בשפת C היא מערך מסוג char. התו האחרון במחרוזת הוא NULL.
`gets(שם-מחרוזת) ;`	**קלט של מחרוזת.**
`puts(שם-מחרוזת) ;`	**פלט של מחרוזת.**
`שם-מחרוזת = itoa(משתנה-שלם) ;`	**המרת מספר שלם למחרוזת** (**i**nt **to a**rray).
`משתנה-שלם = atoi(שם-מחרוזת) ;`	**המרת מחרוזת למספר שלם** (**a**rray **to i**nt).
`משתנה-ממשי = atof(שם-מחרוזת);`	**המרת מחרוזת למספר ממשי** (**a**rray **to f**loat).
`משתנה = strlen(שם-מחרוזת) ;`	**מציאת אורך המחרוזת** (מספר התווים שבה).
`strupr(שם-מחרוזת) ;`	**המרת כל האותיות במחרוזת לגדולות** (ראשיות)
`strlwr(שם-מחרוזת) ;`	**המרת כל האותיות במחרוזת לאותיות קטנות**
`strcmp(מחרוזת-א׳,מחרוזת-ב׳);`	**השוואת שתי מחרוזות.** הפונקציה תחזיר 0 אם המחרוזות זהות; היא תחזיר ערך חיובי אם למחרוזת הראשונה יש ערך לקסיקוגרפי גדול מזה של המחרוזת השנייה, והיא תחזיר ערך שלילי אם למחרוזת השנייה יש ערך לקסיקוגרפי גדול מזה של הראשונה (cat<dog).
`strncmp(מחרוזת-א׳,מחרוזת-ב׳);`	**השוואת n התווים הראשונים בשתי מחרוזות.**
`strcpy (יעד,מקור);`	**העתקת תוכן מחרוזת אחת לשנייה.**
`strncpy (יעד,מקור);`	**העתקת n התווים הראשונים ממחרוזת המקור.**
`strcat (יעד,מקור);`	**שרשור מחרוזת המקור לסוף מחרוזת היעד.**
`strrev (שם-מחרוזת);`	**הפיכת סדר התווים במחרוזת** (תמונת ראי).

דוגמה לקליטת מחרוזות והדפסת אורכה:

```
char st[10];
int i=0;

gets(st);
while (st[i++]);          // עבור לאורך המחרוזות עד התו NULL שבסופה
printf("The length of the string is %d",i-1);
```

או לחילופין:

```
printf("The length of the string is %d",strlen(st));
```

הערה

הפונקציות המתחילות ב-str שייכות ליחידה string.h ולכן, כדי להשתמש בהן יש להכריז על יחידה זו בראש התוכנית.

מבנים ואיגודים

מבנים (רשומות) השמות שאחרי הסוגריים המסולסלים מייצגים משתנים מטיפוס המבנה שהגדרנו. לאחר הגדרת המבנה נוכל להגדיר מבנים נוספים מטיפוס המבנה שהגדרנו, ללא צורך בפירוט חוזר של השדות המרכיבים אותו. נעשה זאת כך: struct שם-מבנה שם , שם... ; פנייה לשדה במבנה תיעשה כך: שם.שם-שדה ניתן לבצע השמה בין שני מבנים ($S_1=S_2$), אך אסור לבצע השוואה בין שני מבנים ($S_1==S_2$).	**struct** שם-מבנה **{** סוג שם-שדה ; סוג שם-שדה ; . . . **}** שם, שם... ;
איגודים איגוד הוא משתנה בודד היכול להכיל ערכים מטיפוסים שונים. גודלו ייקבע על פי סוג השדה הגדול ביותר הכלול בו. ניתן לאחסן באיגוד משתנה אחד בלבד בכל פעם. אופן השימוש באיגוד זהה לאופן השימוש במבנה.	**union** שם-איגוד **{** סוג שם-שדה ; סוג שם-שדה ; . . . **}** שם, שם... ;

דוגמה:

```
struct StudentRec {
         char Name[15];
         int Mark;
                  } Students[40];      // מערך של 40 תלמידים
int i;

for (i=0 ; i<40 ; i++)
{
  gets (Students[i].Name);
  scanf ("%d",&Students[i].Mark);
}
```

1.3 פונקציות

מבנה הפונקציות

מבנה הפונקציה.	טיפוס-מוחזר שם-פונקציה **(** שם סוג, שם סוג,... **)** **{** הגדרות פקודות . . . **return (**ערך**)**; **}**

הטיפוס המוחזר מהפונקציה הוא טיפוס המשתנה, אשר הפונקציה מחזירה בפקודה **return**.

לא כל פונקציה חייבת להחזיר ערך. אם הפונקציה אינה מחזירה ערך, נכתוב במקום הטיפוס המוחזר את המילה **void**. במקרה זה, אי אפשר לכתוב את הפקודה **return**.

בתוך הסוגריים נרשום את רשימת הפרמטרים שהפונקציה תקבל.

שימו לב! הפרמטרים יועברו על פי ערכם (By Value) ולא על פי כתובתם (By Address). כלומר, כל שינוי שהפונקציה תבצע בפרמטרים שקיבלה לא ישנה את המשתנים המקוריים. בסעיף הבא (הדן במצביעים) נלמד כיצד ליצור פונקציה המקבלת פרמטרים By Address.

לא כל פונקציה חייבת לקבל פרמטרים. אם פונקציה אינה מקבלת פרמטרים, נכתוב במקום רשימת הפרמטרים את המילה **void**.

המשתנים שיוגדרו בתוך הפונקציה יהיו מקומיים, והמשמעות היא שיוכרו בפונקציה זו בלבד.

קריאה לפונקציה

כדי לקרוא לפונקציה (להפעיל אותה) עלינו לכתוב את שמה ואחר כך את הערכים, שאנו רוצים להעביר אליה בתוך סוגריים.

אם הפונקציה מחזירה ערך, נוכל לקרוא לה בפקודת הצבה. לדוגמה:

```
שם-משתנה = שם-פונקציה (רשימת משתנים או ערכים מועברים);
```

סדר הערכים שנרשום בתוך הסוגריים צריך להיות זהה לסדר שהגדרנו בפונקציה.

אם הפונקציה אינה מקבלת פרמטרים נכתוב בכל זאת את הסוגריים, אך לא נכתוב ביניהם דבר.

הפונקציה main

בכל תוכנית **חייבת** להיות פונקציה ראשית בעלת השם **main**, אשר ממנה תתחיל ריצת התוכנית. בדרך כלל, פונקציה זו אינה מחזירה או מקבלת ערכים, אך ניתן בכל זאת לעשות זאת.

ניקח כדוגמה ל**תוכנית המקבלת פרמטרים** את התוכנית dir של מערכת ההפעלה DOS. כדי להציג את כל הקבצים בעלי הסיומת exe אנו כותבים בשורת הפקודה של DOS: "dir *.exe". במקרה זה, dir הוא שם התוכנית ו- exe.* היא מחרוזת בקרה שהפונקציה הראשית מקבלת.

כדי ליצור **תוכנית המקבלת פרמטרים**, עלינו לרשום בתוך הסוגריים של הפונקציה main, במקום void, את הפרמטרים **argv** ו-**argc**:

```
void main (int argc, char *argv[])
```



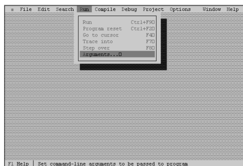
את משמעות הכוכבית הרשומה לפני argv בסוגריים של הפונקציה הראשית, נבין בסעיף הבא הדן במצביעים.

בזמן הפעלת התוכנית, המשתנה argc יכיל את מספר הפרמטרים שנכללו בשורת הפקודה (כולל שם התוכנית), ואילו מערך המחרוזות []argv יכיל את מחרוזות הפרמטרים (בכל תא תישמר המחרוזת של פרמטר אחד).

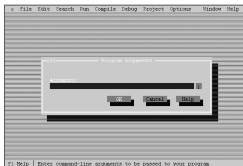
נחזור לדוגמה של התוכנית dir. בתוכנית זו, המשתנה argc יקבל את הערך 2, התא הראשון במערך argv יכיל את המחרוזת "dir" (שם התוכנית), והתא השני במערך argv יכיל את המחרוזת "exe.*".

בפרק 6, תוכלו לראות את המימוש של התוכנית dir ודוגמאות רבות נוספות לתוכניות המקבלות פרמטרים.

כיצד נריץ תוכנית המקבלת פרמטרים מתוך המהדר עצמו? לשם כך נלחץ על התפריט Run, ובו נבחר ב-Arguments (תרשים 1.6). תיפתח תיבת דו-שיח שנכתוב בה את הערכים שאנו רוצים למסור לתוכנית (תרשים 1.7). אחר כך נוכל להריץ את התוכנית כרגיל.



תרשים 1.6



תרשים 1.7

מיקום הפונקציות

יש שתי דרכים לכתוב תוכנית בשפת C. הראשונה היא שיטת "**מטה-מעלה**" (**Bottom-Up**). בשיטה זו כותבים את הפונקציות לפני הפונקציה הראשית (main). כל פונקציה מכירה את הפונקציות הכתובות לפניה ויכולה לקרוא להם. הפונקציה הראשית היא האחרונה (או התחתונה, אם מסתכלים על הפונקציות כמונחות ברבדים, זו על זו) ולכן היא מכירה את כל הפונקציות שקדמו לה. שיטה זו נקראת מטה-מעלה, כי אנו כותבים תחילה את הפונקציות למימוש תהליכי הקצה של היישום, אחר כך את התהליכים המורכבים (המפעילים את תהליכי הקצה שכתבנו מעליהם) ולבסוף כתובה הפונקציה הראשית.

השיטה השנייה היא שיטת "**מעלה-מטה**" (**Top-Down**), שלפיה נכתוב את הפונקציות בסדר הפוך : מהפונקציה הראשית main ועד הפונקציות הבסיסיות ביותר. בשיטה זו הפונקציות קוראות לפונקציות שכתובות אחריהן, ולכן עלינו להכריז על הפונקציות בראש התוכנית כדי שתהיינה מוכרות לפונקציות שנכתבות לפניהן (או מעליהן, אם מסתכלים על הפונקציות כמונחות ברבדים, זו על זו). לשם כך נכתוב בראש התוכנית הכרזה על הפונקציה שאנו רוצים להשתמש בה, ואשר כוללת את הכותרת של הפונקציה ובסופה נקודה-פסיק. הכרזה זו היא אב טיפוס (prototype) על הפונקציה.

יש המעדיפים את השיטה השנייה, בטענה כי היא מקבילה לדרך חשיבתו של אדם סביר, ולא מפתח תוכנה בלבד. בספר זה נכתוב את התוכניות בשיטה הראשונה, והסיבה לכך פשוטה : כאשר מפרקים בעיה גדולה לחלקי משנה ממוקדים, הרבה יותר קל להתמודד איתה. בשיטה זו נכתוב תחילה את הפתרונות של הבעיות הממוקדות, ואחר כך נפתור בעזרתם את הבעיה המורכבת יותר.

לפניכם שתי תוכניות המדגימות את השימוש בפונקציות בשתי השיטות. התוכנית הראשונה מדגימה את השימוש בשיטת מטה-מעלה, והתוכנית השנייה מדגימה את השימוש בשיטת מעלה-מטה.

בספר זה נכתוב את שם התוכנית בשורת ההערה הראשונה. התוכנית שמורה בדיסק, בתיקיה של פרק זה, פרק 1 (או פרק אחר, בהתאמה) תחת התיקיה books\59281.

```
/*** C1E1.C ***/

#include <stdio.h>
#include <conio.h>

int Sum (int a, int b)
{
   return (a+b);
}

void Pelet (void)
{
   printf("3+2=%d",Sum(3,2));
}
```

```
void main (void)
{
   Pelet();
}
```

```
/*** C1E2.C ***/

#include <stdio.h>
#include <conio.h>

void Pelet (void) ;
int Sum (int a, int b) ;

void main (void)
{
   Pelet();
}

void Pelet (void)
{
   printf("3+2=%d",Sum(3,2));
}
int Sum (int a, int b)
{
   return (a+b);
}
```

1.4 טווח הכרה ומשך החיים של משתנים (Scope and Duration)

משתנים גלובליים

- מוגדרים בראש התוכנית (מחוץ לכל הבלוקים).
- מוכרים על ידי כל הפונקציות.
- שומרים על ערכם במשך כל זמן ריצת התוכנית.
- מאותחלים ל-0 אם לא התבצע אתחול מפורש.

משתנים לוקליים (נקראים גם אוטומטיים)

- מוגדרים בתוך פונקציה או בלוק כלשהו.
- מוכרים על ידי הבלוק בו הם מוגדרים בלבד.
- מאבדים את ערכם ונעלמים ביציאה מהבלוק.
- אם קיים גם משתנה גלובלי בעל שם זהה, שינוי ערך הלוקלי בתוך הבלוק לא ישפיע על הגלובלי.
- אינם מאותחלים על ידי המהדר.
- מאוחסנים במחסנית או באוגרים.

משתנים סטטיים - static

- לפני הגדרת המשתנה נכתבת המילה השמורה static.
- משתנה סטטי מוכר על ידי הבלוק בו הוא מוגדר בלבד, אך שומר על ערכו גם מחוץ לבלוק, ולכן כאשר נחזור לבלוק יהיה בו את הערך הקודם.
- אתחול של משתנה סטטי שייכתב בשורת ההגדרה, יתבצע רק פעם אחת.
- מאותחלים ל-0 (פעם אחת בלבד) אם לא התבצע אתחול מפורש.
- הזיכרון מוקצה עבורם לפני ריצת התוכנית ועד לסיומה.
- מאוחסנים בדרך כלל ב-data segments קבועים.

משתני רג'יסטר - register

- לפני הגדרת המשתנה נכתבת המילה השמורה **register**.
- משמעות המילה השמורה register היא **המלצה** למהדר לאחסן משתנה זה באוגר במקום בזיכרון (הגישה לאוגר מהירה מהגישה לזיכרון, ולכן רצוי להשתמש במשתני אוגר למשתנים שערכם משתנה בתדירות גבוהה, כגון מונים).

משתנים הפכפכים - volatile

- לפני הגדרת המשתנה נכתבת המילה השמורה **volatile**.
- משמעות המילה השמורה volatile היא **דרישה** ממהדר לאחסן משתנה זה בזיכרון ולא באוגר זמני.
- מיועד למשתנים הפכפכים אשר עלולים להשתנות על ידי פסיקה, תוכנית אחרת או דרייבר חיצוני (אם המשתנה היה שמור באוגר זמני, התוכנית החיצונית לא היתה יכולה לגשת אליו).

הערה: בנוסף לרשימה זו, קיימים סוגים נוספים של משתנים עליהם נדבר בהמשך הספר.

1.5 מצביעים והקצאות זיכרון

עבודה עם מצביעים רגילים

הגדרת מצביע	סוג *****שם-מצביע ;

את העבודה עם מצביעים בשפת C ניתן לסכם בשתי נוסחאות:

& = **הכתובת של...**

* = **הערך המוצבע ע"י...**

דוגמה הממחישה את שתי הנוסחאות:

```
int x, y, *p1, *p2 ;

p1 = &x   ;          // x המצביע p1 שווה לכתובת של
*p1 = 4   ;          // 4 = p1 הערך המוצבע ע"י
p2 = &y   ;          // y המצביע p2 שווה לכתובת של
*p2 = *p1 ;          // p1 הערך המוצבע ע"י p2 שווה לערך המוצבע ע"י
```

בסיום ביצוע קטע הקוד: x=y=*p1=*p2=4 נקבל: p1 מצביע על x, p2 מצביע על y.

העברת פרמטרים By Address

כפי שראינו בסעיף 1.3, כאשר אנו מעבירים פרמטרים לפונקציה, הם מועברים על פי ערכם בלבד (by value). על כן, אם נשנה בתוך הפונקציה את ערכם, המשתנים המקוריים לא ישתנו.

כדי ליצור פונקציה היכולה לשנות את המשתנים המועברים אליה, עלינו להעביר אותם על פי כתובתם (by address).

נעשה זאת על ידי שימוש במצביעים, בכותרת הפונקציה נגדיר את הפרמטרים כמצביעים. בכל מקום בו נרצה להשתמש בערכם של הפרמטרים נוסיף את הסימן *, ובקריאה לפונקציה נעביר את המשתנים על פי כתובתם באמצעות הסימן &.

זו הסיבה שבגללה בכל פקודה, המשנה את המשתנים המועברים אליה, כתבנו את הסימן & (כמו למשל ב-scanf).

דוגמה לתוכנית המורכבת מפונקציה המקבלת שני משתנים לפי כתובת (By Address) ומחליפה ביניהם:

```
#include <stdio.h>

void Switch (int *a, int*b)
{
   int temp ;
   temp = *a ;
   *a = *b ;
   *b = temp ;
}

void main (void)
{
   int x,y;
   scanf("%d %d",&x,&y);
   Switch(&x,&y);
   printf("%d %d",x,y);
}
```

מצביעים למערכים ומחרוזות

ההגדרה המפורטת למערך בשפת C היא:

"מערך הוא אוסף רציף של נתונים מסוג זהה. שם המערך הוא מצביע קבוע לאיבר הראשון בו".

מהגדרה זו אנו לומדים כי:

ArrayName = &ArrayName[0]

ArrayName[i] = (*(ArrayName+i))

ArrayName[i][j] = (*(*(ArrayName+i))+j)

מכאן אנו יכולים להבין כי:

- בכל פעם שהשתמשנו במערכים ומחרוזות, השתמשנו במצביעים מבלי לדעת זאת.
- כל פונקציה המקבלת מערך או מחרוזת, מקבלת אותו לפי כתובת.

זאת הסיבה לכך, שכאשר קראנו לפונקציות המשנות את ערך המערך (כמו למשל הפונקציה gets) לא השתמשנו בסימן &.

אם נרצה לכתוב פונקציה המקבלת מחרוזת כמצביע, נגדיר ברשימת הפרמטרים מצביע מסוג char. כדי לדעת אם הגענו לסוף המחרוזת, נבדוק אם התו הנוכחי הוא NULL.

אם נרצה לכתוב פונקציה המקבלת מערך כמצביע, נהיה חייבים להעביר אליה גם ערך המציין את מספר האיברים במערך, כדי לדעת מתי מגיעים לסופו.

מצביעים למבנים ואיגודים

כשמעבירים מבנים ואיגודים אל פונקציה, הם מועברים על פי ערכם (כמו כל נתון פשוט אחר). הפונקציה יוצרת מבנה מקומי ומעתיקה לתוכו את תוכן המבנה המועבר אליה. לכן, כשנרצה לכתוב פונקציה המקבלת מבנה לפי כתובת (By Address), נעביר את המבנה באמצעות מצביעים, כמו בשיטת העברת הפרמטרים הרגילה שנעשית לפי כתובת.

כדי לשנות ערך של שדה במבנה שמצביע פונה אליו, נכתוב:

(*שם-מצביע).שם-שדה = ערך ;

יש דרך קצרה לעשות זאת. במקום להשתמש בסימן * ובסוגריים, נוכל להשתמש בחץ -> (משמאל, סימן "מינוס" ואחריו סימן "גדול מ-"). נעשה זאת כך:

שם-מצביע -> שם-שדה = ערך ;

דוגמה מסכמת:

```
/*** C1E3.C ***/
#include <stdio.h>

struct TalRec {
         char Name[10];
         int Mark ;
             };

void PrintRec (struct TalRec T)
{
   puts(T.Name);
   printf("%d",T.Mark);
}

void GetRec (struct TalRec *Tp)
{
   gets (Tp -> Name) ;
   scanf("%d",&Tp->Mark);
}

void main (void)
{
   struct TalRec Talmid;

   GetRec (&Talmid);
   PrintRec(Talmid);
}
```

הקצאת זיכרון דינמית

כדי להשתמש בפונקציות הקצאת הזיכרון יש לקרוא ליחידת הספריה <alloc.h>!

הקצאת בלוק זיכרון.	שם-מצביע = (סוג *)**malloc** (גודל);
הקצאת בלוק זיכרון ואיפוסו.	שם-מצביע = (סוג *)**calloc** (גודל,כמות);
הקצאה מחודשת של בלוק זיכרון. (תוכן הבלוק הישן מועתק לחדש)	שם-מצביע = (סוג *)**realloc** (גודל,שם מצביע);
שחרור בלוק זיכרון שהוקצה.	**free** (שם-מצביע);

את גודל הבלוק נוכל לחשב בעזרת הפונקציה "sizeof (סוג)", כפי שעשינו בסעיפים הקודמים. שימו לב, שהשתמשנו כאן ב**הסבה בכוח** לסוג המצביע הרצוי (ראה סעיף 1.1). עשינו זאת מפני שפונקציות ההקצאה מחזירות מצביע כללי מסוג void, המצביע על תחילת בלוק הזיכרון שהוקצה. אם ההקצאה נכשלת (עקב מחסור בזיכרון), המצביע מקבל את הערך **NULL**.

חובה לשחרר את בלוק הזיכרון שהוקצה בתום השימוש בו, אחרת התוכנית עלולה להיכשל בשל מחסור בשטח זיכרון פנוי לעבודה.

דוגמה להקצאת זיכרון עבור מערך של 5 מספרים שלמים:

```
int *p ;
// הקצאת בלוק מסוג שלם בגודל חמישה שלמים. אם ההקצאה תיכשל תודפס הודעה
if (!p=(int *)malloc(5*sizeof(int)) )
{
      printf ("אין מספיק זיכרון\n");
}
```

1.6 קבצים

הגדרה, פתיחה וסגירה

הגדרת מצביע לקובץ.	**FILE** *שם-לוגי ;
פתיחת קובץ. (רשימת מודים בתרשים 1.8)	שם-לוגי = **fopen**("שם-פיסי","מוד");
בדיקה האם הגענו לסוף הקובץ.	**feof**(שם-לוגי);
סגירת קובץ.	**fclose** (שם-לוגי);
סגירת כל הקבצים הפתוחים.	**fcloseall**();

שימו לב, המילה FILE מופיעה באותיות גדולות!

השם הפיסי יורכב משם הקובץ (כולל סיומת) ומיקומו. אם הקובץ נמצא בתיקיה (ספריה) שממנה אנו מריצים את התוכנית, אין צורך לכתוב את מיקומו.

כדי "לרוץ" על כל הקובץ עד סופו, נכתוב:

```
while (!feof(LogiName))
```

מוד טקסט	מוד בינארי	משמעות
"r"	"rb"	פתח קובץ קיים לקריאה בלבד.
"w"	"wb"	פתח קובץ חדש לכתיבה בלבד (אם קיים, נמחק ונבנה מחדש).
"a"	"ab"	פתח קובץ קיים לכתיבה בסופו (אם לא קיים נבנה).
"r+"	"r+b"	פתח קובץ קיים לקריאה וכתיבה
"w+"	"w+b"	פתח קובץ חדש לכתיבה וקריאה
"a+"	"a+b"	פתח קובץ קיים לקריאה וכתיבה בסופו (אם לא קיים נבנה)

תרשים 1.8

קבצי טקסט

קריאת תו אחד מתוך קובץ לתוך משתנה מסוג תו.	`שם-משתנה = fgetc (שם-לוגי) ;`
כתיבת תו אחד לתוך קובץ.	`fputc(שם-משתנה, שם-לוגי) ;`
קריאת שורה שלמה מתוך הקובץ אל מחרוזת. התווים נקלטים במחרוזת עד לסימן המסמל סוף שורה, או עד "אורך" המחרוזת בתווים, כפי שהוגדר בפקודה.	`fgets(שם-מחרוזת, אורך, שם-לוגי) ;`
כתיבת מחרוזת לתוך קובץ.	`fputs(שם-מחרוזת, שם-לוגי) ;`

התו האחרון בקובץ הוא "**EOF**".

כתיבת נתונים שונים בקובץ טקסט

קריאת נתון מקובץ טקסט.	`fscanf(שם-לוגי, "תו-בקרה", &משתנה) ;`
כתיבת נתון לקובץ טקסט.	`fprintf(שם-לוגי, "תו-בקרה", משתנה) ;`

רבים טועים לחשוב שהפקודות fscanf ו-fprintf מיועדות לקבצים בינאריים. פקודות אלו שקולות לפקודות scanf ו-printf שמבצעות המרה בין הטקסט שעל המסך לבין משתנים מסוגים שונים. לכן, גם הפקודות fscanf ו-fprintf מתייחסות לקובץ כ-streem של טקסט ומבצעות המרה בין המשתנים לבין הטקסט שהן כותבות או קוראות מהקובץ.

לדוגמה, אם נכתוב לקובץ בינארי משתנה מסוג int שמכיל את הערך 1000 הוא יתפוס שני בתים בקובץ (גודל של int), אך אם נכתוב אותו בעזרת fprintf לקובץ, הוא יתפוס ארבעה בתים (ארבע תווים בגודל בית).

ראו רשימת תווי בקרה בתרשים 1.3.

נתונים פשוטים בקובץ בינארי

קריאת נתון מקובץ בינארי.	`fread(&שם-לוגי, 1, sizeof(סוג), שם-לוגי);`
כתיבת נתון לקובץ בינארי.	`fwrite(&שם-לוגי, 1, sizeof(סוג), שם-לוגי);`

מערכים בקובץ בינארי

קריאת מערך מקובץ בינארי.	`fread(שם-מערך, sizeof(סוג), כמות, שם-לוגי);`
כתיבת מערך לקובץ בינארי.	`fwrite(שם-מערך, sizeof(סוג), כמות, שם-לוגי);`

סוג – סוג האיברים שבמערך (תרשים 1.1).

כמות – מספר האיברים שבמערך.

שימו לב שהפעם, אין צורך לכתוב את הסימן & (ראה סעיף 1.4).

דוגמה לקליטת נתונים מקובץ אל מערך, איפוס האיבר השישי וכתיבת המערך לקובץ אחר:

```
int Vec[10] ;

fread (Vec,sizeof(int),10,File1) ;
Vec[5] = 0;
fwrite(Vec,sizeof(int),10,File2);
```

מבנים (רשומות) בקובץ בינארי

קריאת מבנה מקובץ בינארי.	`fread(&שם-מבנה, sizeof(סוג), כמות, שם-לוגי);`
כתיבת מבנה לקובץ בינארי.	`fwrite(&שם-מבנה, sizeof(סוג), כמות, שם-לוגי);`

סוג מתייחס לטיפוס המבנה (שם-מבנה struct).

שימו לב שהפעם, גם בקריאה וגם בכתיבה, יש לכתוב את הסימן &. הסיבה לכך היא שהפונקציות האלו מתאימות גם למערכים (כמו שראינו קודם), ולכן עליהן לקבל את הפרמטר לפי כתובת (By Address. ראו סעיף 1.4).

דוגמה:

```
struct TalRec                  // הגדרת רשומת תלמיד
{
   char FirstName[10];
   char LastName[10];
   int Mark;
} Talmid , Talmidim[30];

Talmid.FirstName = "דני";
Talmid.LastName = "לוי";
Talmid.Mark = 100;

                                  // כתיבת רשומת התלמיד לקובץ
fwrite (&Talmid,sizeof(struct TalRec),1,File1);

                                  // קריאת רשומות של 30 תלמידים למערך התלמידים
fread(Talmidim,sizeof(struct TalRec),30,File2);
```

תזוזה בתוך קובץ (גישה ישירה)

קפיצה למקום כלשהו בקובץ.	`fseek(שם-לוגי,כמות*sizeof(סוג),מילת בקרה);`

כמות – מספר הרשומות או המשתנים שאנו רוצים לדלג עליהם. אם נרצה לדלג לאחור, הערך יהיה שלילי.

מילת הבקרה מציינת מאיזו נקודה בקובץ אנו רוצים לקפוץ (תרשים 1.9).

מילת בקרה	משמעות
SEEK_SET	קפוץ מתחילת הקובץ. (הכמות חיובית)
SEEK_END	קפוץ אחורה מסוף הקובץ. (הכמות שלילית)
SEEK_CUR	קפוץ מהמקום הנוכחי בקובץ. (הכמות יכולה להיות חיובית או שלילית)

תרשים 1.9

כדי להגיע לנתון האחרון בקובץ (שלפני הערך EOF), נדלג נתון אחד לאחור ממיקום סוף הקובץ:

```
fseek (שם-לוגי,-1*sizeof(סוג),SEEK_END);
```

וכדי לעבור ל"נתון" הראשון בקובץ, נקפוץ "אפס" נתונים מתחילת הקובץ:

```
fseek(שם-לוגי,0,SEEK_SET) ;
```

הערה

״נתון״ יכול להיות תו, רשומה, מערך, או כל ערך אחר. זאת מפני שגודל הקפיצה נקבע על פי מכפלת (סוג הנתון)sizeof בכמות.

מיקום המצביע בקובץ.	משתנה-ארוך = **ftell (שם-לוגי)** ;

שימו לב, שהמיקום המוחזר הוא מסוג משתנה ארוך (long).

כדי לדעת מהו מספר הנתון הנוכחי בקובץ (שמצביע הקובץ עומד עליו), נחלק את המיקום בגודל נתון אחד:

מספר הרשומה או הנתון מתחילת הקובץ = ftell (שם-לוגי) / sizeof (סוג) ;

דוגמה להעתקת הציון של התלמיד החמישי אל התלמיד הראשון ולתלמיד האחרון:

```
FILE *Class1 ;
struct Tal_rec
     {
       char Fname[10] ;
       char Lname[10] ;
       int Mark ;
     } Talmid ;
int TmpMarkm ;

Class1 = fopen ("c:\\Marks.dat","r+b" ) ;
                              // שמירת הציון של התלמיד החמישי
fseek (Class1,4*sizeof(struct Tal_Rec),SEEK_SET);
fread  ( &Talmid , sizeof(struct Tal_rec),1,class1 ) ;
TmpMark = Talmid .Mark ;
                              // תיקון התלמיד הראשון
fseek(Class1,0,SEEK_SET) ;
fread(&Talmid , sizeof(struct Tal_rec),1,class1 ) ;
Talmid.Mark = TmpMark ;
fseek(Class1,-1*sizeof(struct Tal_Rec),SEEK_CUR);
fwrite(&Talmid , sizeof(struct Tal_rec),1,class1 ) ;
                              // תיקון התלמיד האחרון בקובץ
fseek (Class1,-1*sizeof(struct Tal_Rec),SEEK_END);
fread (&Talmid , sizeof(struct Tal_rec),1,class1 ) ;
Talmid.Mark = TmpMark ;
fseek (Class1,-1*sizeof(struct Tal_Rec),SEEK_CUR);
fwrite ( &Talmid , sizeof(struct Tal_rec),1,class1 ) ;

fclose ( class1 ) ;
```

פרק 2

עבודה בסיביות

בפרק זה נלמד פעולות שונות שניתן לבצע על סיביות.

- ✓ **נכיר את שיטות הספירה השונות.**
- ✓ **נלמד את הפעולות שניתן לבצע על סיביות.**
- ✓ **נכיר מספר שימושים לעבודה בסיביות.**

אני מניח שכבר נתקלתם בעבר בעבודה בסיביות. נושא זה פשוט, והסיבה שבגללה לא כללתי אותו בפרק הקודם והקדשתי לו פרק נפרד היא, שברוב הספרים האחרים ממעיטים בערך נושא זה (כי אינו נחוץ בתוכניות פשוטות). בספר זה תגלו, כי כמעט בכל פרק נצטרך לבצע פעולות שונות על סיביות ולכן חשוב להכיר נושא זה היטב.

2.1 שיטות ספירה

שפת C מאפשרת לייצג ערכים בשיטות הספירה העשרונית, האוקטלית וההקסדצימלית.

מספר עשרוני (בסיס 10) מיוצג על ידי מספר רגיל שאינו מתחיל באפס.

מספר אוקטלי (בסיס 8) מיוצג על ידי מספר המתחיל בספרה אפס, לדוגמה 012 (= 12(8)).

מספר הקסדצימלי (בסיס 16) מיוצג על ידי מספר שלפניו "0x" , לדוגמה 0x12 (גם 12h).
שים לב: "0" הוא "אפס".

כדי להציג ערך משתנה בייצוג אוקטלי או הקסדצימלי, נשתמש בתו בקרה מתאים בפקודת הפלט (ראה תרשים 1.3 בפרק 1).

ברוב המקרים נשתמש בייצוג העשרוני הרגיל, אך כשנרצה לפנות לרכיב חומרה, או כשנעבוד עם פסיקות, ניאלץ להשתמש בשתי שיטות הייצוג האחרות.

עד כאן הכל פשוט וברור, הבעיה מתחילה כשרוצים לייצג ערכים בינאריים. בשפת C לא ניתן לייצג ערכים בייצוג בינארי, ולכן בכל פעם שנרצה להשתמש בערכים בינאריים ניאלץ להציגם כערכים הקסדצימליים.

ניתן לייצג כל ארבע ספרות בינאריות בספרה הקסדצימלית באופן הבא: נכפול את ערך הספרה הראשונה מימין ב-1, את ערך הספרה השנייה מימין ב-2, את השלישית ב-4 ואת הרביעית ב-8 (כלומר,בחזקות עולות של 2). נחבר את ארבע המכפלות ונקבל את ערך המספר הבינארי.

לדוגמה: אם נרצה לייצג את ערכו של המספר הבינארי 1010 בייצוג הקסדצימלי, נחשב ונמצא תחילה את ערכו בשיטה העשרונית: 2*1 + 8*1 = 10, ידוע כי 10 בבסיס 10 שווה ל- aH בייצוג הקסדצימלי.

אם כך, בכל פעם שנרצה לייצג ערך בינארי בשפת C, נחלק אותו לחלקים בני ארבע ספרות וכל ארבע ספרות נמיר לספרה הקסדצימלית. החלוקה מתחילה מימין!

לדוגמה: ניקח את המספר הבינארי 111111. ארבע הספרות הראשונות שלו מימין מייצגות את הספרה ההקסדצימלית f, ושתי הספרות הבאות משמאל שוות למספר הבינארי 0011 ששווה לספרה ההקסדצימלית 3. לכן 3fH = 111111(2).

2.2 בית, מילה ומילה כפולה

ארבעת טיפוסי המשתנים הנפוצים ביותר לאחסון נתונים הם:

סיבית (bit) היא היחידה הקטנה ביותר לייצוג נתונים במחשב, ומציינת ספרה בינארית שערכה "1" או "0". כשערך הסיבית הוא "1" ניתן לומר שערכה "אמת", וכשערכה "0" ניתן לומר שערכה הוא "שקר".

בית (byte) הוא יחידת ייצוג נתונים בגודל שמונה סיביות (שתי ספרות הקסדצימליות).

מילה (word) היא יחידת ייצוג נתונים בגודל שני בתים (16 סיביות, או ארבע ספרות הקסדצימליות).

מילה כפולה (dword) היא יחידת ייצוג נתונים בגודל ארבעה בתים (32 סיביות, או שמונה ספרות הקסדצימליות).

בפרק הקודם הכרנו את סוגי המשתנים הרגילים (int, Short, long וכו') וציינו את גודלם בבתים (תרשים 2.1). מכיון שייצוג הסימן של כל ערך (שלילי או חיובי) נקבע על ידי הסיבית השמאלית ביותר שלו, הערך החיובי המקסימלי שניתן להציב במשתנה כזה הוא המספר הגדול ביותר, שניתן לייצג על ידי מספר הסיביות של המשתנה פחות אחד.

מסיבה זו (וסיבות נוספות שיוסברו בהמשך), כשעובדים עם סיביות נהוג לעבוד עם סוגי משתנים חיוביים (**unsigned**). כך נתייחס למשתנה כרצף של סיביות לוגיות ללא התחשבות בערך הממשי של המשתנה. כלומר, אנו מתייחסים לערך של כל הסיביות כערך חיובי.

שפת C מאפשרת להגדיר טיפוסי משתנים חדשים באמצעות שילוב של טיפוסים קיימים. הדבר מתבצע בעזרת מנגנון הגדרה הנקרא **typedef**.

ההגדרה מתבצעת באופן הבא:

```
typedef שילוב טיפוסים קיימים שם הטיפוס החדש;
```

לאחר שמגדירים טיפוס משתנה חדש ניתן להגדיר בכל מקום בתוכנית משתנה מסוג טיפוס זה, כאילו הוא אחד משאר סוגי המשתנים הרגילים.

אם כך, בכל פעם שנצטרך לבצע פעולות שונות על סיביות, נגדיר בראש התוכנית את הטיפוסים האלה:

```
typedef unsigned char byte;
typedef unsigned short word;
typedef unsigned long dword;
```

שימו לב! בכל מקום בספר זה (וגם בספרים ובתוכניות אחרות לעיתים), כשנגדיר משתנה מאחד הטיפוסים האלה, זכרו שיש לרשום את שלוש ההגדרות בראש התוכנית. מוזר, אך למרות שהשימוש בטיפוסים אלה נפוץ מאוד כשעוסקים בסיביות, הם אינם מוכרים למהדר ויש להגדירם במפורש.

2.3 פעולות לוגיות בסיביות

ניתן לבצע על סיביות ארבע פעולות לוגיות:

1. פעולת **NOT**, שסימונה בשפת C הוא "~". פעולה זו הופכת את ערך הסיבית מ"אמת" ל"שקר" ולהפך.
2. פעולת **AND**, שסימונה בשפת C הוא "&". פעולה זו מבצעת כפל לוגי בין שתי סיביות (התוצאה תהיה אמת, רק אם ערך שתי הסיביות הוא "אמת").
3. פעולת **OR**, שסימונה "|". פעולה זו מבצעת חיבור לוגי בין שתי סיביות (מספיק שערך של אחת משני הסיביות יהיה "אמת", כדי שהתוצאה תהיה "אמת").
4. פעולת **XOR** שסימונה "^". פעולה זו מבצעת חיבור לוגי אקסקלוסיבי בין שתי סיביות (כמו החיבור הלוגי הרגיל, אלא שהפעם רק ערך אחת הסיביות צריך להיות "אמת", כדי שהתוצאה תהיה "אמת").

XOR

^	0	1
0	0	1
1	1	0

OR

|	0	1
0	0	1
1	1	1

AND

&	0	1
0	0	0
1	0	1

כאשר מבצעים אחת מהפעולות הללו על משתנים, הפעולה מתבצעת בין כל שתי סיביות מתאימות של המשתנים (**bitwise**).

לדוגמה:

```
x=0x5a;       y=0xa5;       z=x|y;

x = aH = 01011010
y = 5H = 10100101
     z = 11111111 = ffH
```

2.4 מסכות

מסכה (**mask**) היא צירוף של סיביות בינאריות. משתמשים במסכה כדי לבצע פעולות שונות על חלק מהסיביות של משתנה כלשהו. אחד השימושים הנפוצים ביותר למסכות הוא בדיקת ערך של סיבית מסוימת במשתנה.

נניח לדוגמה, שאנו רוצים לבדוק האם הסיבית השלישית במשתנה מסוים מורמת (ערך הסיבית b2 הוא "1", או "אמת"). לשם כך, נכין מסכה שבה נציב "1" בסיבית השלישית ו-"0" בשאר הסיביות. המסכה תיראה כך: <00000100> וערכה יהיה 4H.



שימו לב שסימון הסיביות הוא bn, כאשר n הוא ערך מ-0 עד מספר הסיביות האפשרי במשתנה. הספרה הראשונה מימין היא 0.

התיאור "מורמת" שאנו מייחסים לסיבית פירושו שערכה "1" (on). סיבית "מורדת" ערכה הוא "0" (off). לעיתים אומרים גם "דלוקה" או "כבויה", בהתאמה.

כשנבצע כעת את הפעולה AND על המשתנה שאנו רוצים לבדוק, עם המסכה שיצרנו, הפעולה AND תחזיר "1" בסיבית השלישית אם הסיבית השלישית במשתנה מורמת. בשאר הסיביות הפעולה תחזיר אפס, בלי להתחשב למצב הסיביות במשתנה (כדי שהפעולה AND תחזיר אמת, שתי הסיביות שהיא בודקת צריכות להיות מורמות; ואם אחת אינה מורמת, אין כל חשיבות למצבה של השנייה).

הערך שיוחזר מהפעולה יהיה אפס, אם הסיבית השלישית במשתנה אינה מורמת, ו-4 אם היא מורמת. מכיוון שבשפת C אפס נחשב כשקר וכל ערך אחר נחשב כאמת, ניתן לכתוב זאת כך:

```
if ( x & 0x04 )
   printf("הסיבית השלישית מורמת");
else
   printf("הסיבית השלישית אינה מורמת");
```

באופן זה נוכל לבדוק את ערכה של כל אחת מהסיביות במשתנה.

כדי להקל עלינו במציאת המסכה המתאימה, נכין מראש טבלה של המסכות לבדיקת סיבית בודדת במשתנה מסוג בית:

בדיקת הסיבית הראשונה	b0	1H = 00000001
בדיקת הסיבית השנייה	b1	2H = 00000010
בדיקת הסיבית השלישית	b2	4H = 00000100
בדיקת הסיבית הרביעית	b3	8H = 00001000
בדיקת הסיבית החמישית	b4	10H = 00010000
בדיקת הסיבית השישית	b5	20H = 00100000
בדיקת הסיבית השביעית	b6	40H = 01000000
בדיקת הסיבית השמינית	b7	80H = 10000000



לעיתים נרצה לאלץ סיבית מסוימת במשתנה להיות "דלוקה". לשם כך נבצע פעולת OR בין המשתנה לבין המסכה המתאימה.

2.5 הזזת סיביות (כפל/חילוק מקוצר)

כפל על ידי הזזה שמאלה

הזזת סיביות שמאלה נעשית באמצעות האופרטור "<<". בפעולה זו, הסיביות שבאופרנד השמאלי זזות שמאלה מספר פעמים השווה לערכו של האופרנד הימני. הסיביות ש"נושרות" מהגבול השמאלי של המשתנה הולכות לאיבוד, ומימין "נכנסות" סיביות כבויות (שערכן אפס).

ניקח כדוגמה את הפקודה הבאה, המזיזה פעמיים את סיביות המשתנה X שערכו a3H:

```
y = x<<2 ;
          x = 23H = 00100011        <- מצב התחלתי
              46H = 01000110        <- לאחר הזזה אחת
          y = 8cH = 10001100        <- לאחר שני הזזות
```

בסיום ביצוע הפעולה, הערך של Y יהיה 8cH והערך של X יישאר a3H.

להזזת הסיביות יש משמעות נוספת. נסו להבחין מה קורה לערך מבחינה מתמטית לאחר כל הזזה. הבה נבחן פעולה זו.

נמיר את שלושת הערכים ההקסדצימליים שקיבלנו בתרגיל הקודם לערכים עשרוניים, כדי שנוכל לראות זאת יותר ברור. במצב ההתחלתי היה הערך 23H שווה ל-35, אחרי הזזה אחת הפך הערך ל-46H ששווה ל-70, אחרי הזזה נוספת הפך הערך ל-8cH ששווה ל-140. שימו לב לערכים: 35, 70, 140. בכל פעם שהזזנו את הסיביות, הערך נכפל פי 2.

מסקנה: הזזת הסיביות שמאלה מבצעת כפל מקוצר בחזקה של 2, על פי מספר ההזזות.

לדוגמה, 3 הזזות שקולות לכפל ב-2 בחזקת 3, או כפל ב-8.

מכיון שפעולת כפל זו מהירה בהרבה מפעולת הכפל הרגילה (המתבצעת על ידי אוסף של פעולות חיבור), מכנים אותה בשם **כפל מקוצר**.

חילוק על ידי הזזה ימינה

באופן דומה לכפל, ניתן גם לחלק מספר בינארי בשיטת ההזזה. הפעם נזיז את הסיביות של המשתנה ימינה באמצעות האופרטור ">>". בפעולה זו, הסיביות ש"נושרות" מהגבול הימני של המשתנה הולכות לאיבוד, ומהצד השמאלי נכנסות סיביות "1" או "0" באופן הבא:

- אם ערך המשתנה מזוהה כחיובי, שיש לו סיבית "0" שמאלית, מוכנסת סיבית "0".
- אם ערך המשתנה מזוהה כשלילי, שיש לו סיבית "1" שמאלית, מוכנסת סיבית "1".
- אם ערך המשתנה מזוהה כחסר סימן (unsign), תמיד תיכנסנה סיביות שערכן "0".

כדי למנוע טעויות, עלינו לדעת שאם נשתמש באחד הטיפוסים חסרי הסימן שהגדרנו (byte , dword , word), הסיביות שתיכנסנה משמאל תהיינה תמיד בעלות ערך "0".

ניקח כדוגמה את הפקודה הבאה, המזיזה פעמיים את סיביות המשתנה X שערכו 14H:

```
y = x>>2 ;
             x = 14H = 00010100        <- מצב התחלתי
                  aH = 00001010        <- לאחר הזזה אחת
             y =  5H = 00000101        <- לאחר שתי הזזות
```

גם הפעם יש משמעות מתמטית להזזת הסיביות. נמיר את שלושת הערכים ונקבל: 20, 10, 5. הפעם, בהזזה ימינה, כל הזזה מחלקת את הערך ב-2.

מסקנה: פעולת הזזת הסיביות ימינה מבצעת חילוק מקוצר בחזקה של 2.

לדוגמה, 3 הזזות ימינה שקולות לחלוקה ב-2 בחזקת 3, כלומר לחלוקה ב-8.

מכיון שחילוק בדרך זו מהיר בהרבה מפעולת החילוק הרגילה (המתבצעת על ידי אוסף פעולות חיסור) נקראת הזזה זו בשם **חילוק מקוצר**.

2.6 הדפסת ערך בינארי של משתנה

הידע שברשותנו מאפשר לנו כעת להדפיס את ערכו הבינארי של משתנה. הדרך הפשוטה ביותר (אך ארוכה ולא יעילה) היא להעביר את המשתנה דרך שמונה המסכות שהכנו. כל מסכה תבדוק ערך של סיבית אחרת במשתנה, וכך נדע להדפיס את הספרות "1" או "0" כנדרש. שיטה יותר יעילה היא לבדוק באמצעות המסכה את הסיבית הראשונה (b0), להדפיס, להזיז את הסיביות של המשתנה ימינה ושוב לבדוק את ערך הסיבית הימנית ביותר.

כדי לפעול בדרך זו ניצור תחילה פונקציה המקבלת משתנה מסוג בית, ומשתנה נוסף המציין את מיקום הסיבית בין אפס (ל-b0) עד שבע (ל-b7). הפונקציה תזיז את סיביות המשתנה ימינה כמספר הסיבית לבדיקה, תעביר את הערך שיתקבל במסכה לבדיקת הסיבית הראשונה (b0), ותדפיס אחד או אפס בהתאם.

שימו לב, כשרוצים להזיז את סיביות המשתנה עצמו, ניתן להשתמש בהצבה מקוצרת (תזכורת: כשרוצים להציב במשתנה ביצוע פעולה כלשהי עליו, כותבים את המשתנה, אחריו את האופרטור הרצוי, אחריו סימן שווה ואחריו את האופרנד השני).

עכשיו נותר לנו רק לכתוב לולאה שעבור כל סיבית תפעיל את הפונקציה לבדיקת סיבית. שימו לב, שמכיון שהפלט מודפס משמאל לימין, הלולאה פועלת מהסיבית האחרונה אל הראשונה, בכיוון שמאל לימין.

התוכנית תיראה כך:

```
/*** binary.c ***/

#include <stdio.h>
typedef unsigned char byte;

void TestBit (byte X, int Place)
{
   X >>= Place;
   X &= 0x1;

   if (X)
      printf("1");
   else
      printf("0");
}

void main (void)
{
   byte Num = 0x55;   // הכנס כל ערך הקסדצימלי דו ספרתי שתרצה לבדוק
   int i;

   for (i=7;i>=0;i--)
      TestBit(Num,i);
}
```

2.7 XOR כבדיקת זוגיות סיביות

לפעולה XOR יש שימוש נפוץ נוסף והוא בדיקת זוגיות סיביות.

ניקח לדוגמה את הערך 17H ונבצע פעולת XOR בין כל הסיביות המרכיבות אותו (בכל פעם ניקח זוג סיביות ונבצע ביניהם פעולת XOR).

```
17H = 00010111        0^0^0^1^0^1^1^1 = ?
                        0^0^1^0^1^1^1
                          0^1^0^1^1^1
                            1^0^1^1^1
                              1^1^1^1
                                0^1^1
                                  1^1 = 0
```

עכשיו נסו לחזור על פעולה זו עבור הערך 16H. התשובה שתקבלו צריכה להיות "1".

השוני בתשובה אינו מקרי. **פעולת XOR המופעלת על שרשרת סיביות מחזירה אפס אם מספר הסיביות המורמות בשרשרת הוא זוגי, ואחד אם מספר הסיביות המורמות בשרשרת הוא אי-זוגי.**

לתופעה זו ישנם שימושים רבים. בהמשך נכיר שניים מהם.

סיבית בדיקת זוגיות בתקשורת נתונים

בתקשורת נתונים בין מחשבים משודרים נתונים ממחשב אחד לאחר. לעיתים, עקב רעשים בקו או עקב הפרעות אחרות, הנתונים שמתקבלים בצד הקולט אינם זהים לנתונים שנשלחו מהצד המשדר. כיון שאפילו שגיאה בסיבית אחת עלולה להיות מסוכנת מאוד (לדוגמה, כאשר הנתונים המועברים הם סכומי כסף), יש צורך לבדוק תמיד את הנתונים המתקבלים. אם מתגלית שגיאה, צריך לבקש מהצד המשדר לחזור על הפעולה ולשדר שוב את גוש הנתונים שנפגם.

כיצד נדע אם התרחשה שגיאה בקטע הנתונים שהתקבל? לשם כך, מחלקים את הנתונים ליחידות קטנות הנקראות **מסגרות**, ולכל אחת מהן מוסיפים שדה המיועד לבדיקת שגיאות.

אחת מבדיקות גילוי השגיאות הפשוטה והנפוצה ביותר היא **בדיקת הזוגיות** (**parity check**). בשיטה זו מוסיפים למסגרת סיבית זוגיות. אם מספר הסיביות המורמות במסגרת הוא זוגי נוסיף סיבית שערכה אפס, ואם מספר הסיביות המורמות הוא אי-זוגי נוסיף סיבית שערכה אחד.

התחנה המשדרת משדרת את מסגרת הנתונים ומוסיפה לה את סיבית הזוגיות המתאימה, התחנה הקולטת מקבלת את המסגרת ובודקת האם ערך סיבית הזוגיות אמנם מתאים למסגרת הנתונים שהתקבלה. אם יש אי התאמה וסיבית הזוגיות שהתקבלה אינה מתאימה לתוצאת בדיקת הסיביות שנקלטו, התחנה הקולטת מודיעה לתחנה המשדרת על שגיאה ומבקשת לשלוח שוב את מסגרת הנתונים הפגומה.

כיצד נוכל לבדוק מה צריך להיות ערך סיבית הזוגיות? נוכל לגלות זאת בקלות על ידי סדרה של פעולות XOR בין כל סיביות המסגרת. כך, כמו שהסברנו בתחילת סעיף זה, הערך שיתקבל ישלים את מספר הסיביות המורמות שבמסגרת לערך זוגי, שהוא ערך סיבית הזוגיות.



שיטת בדיקת הזוגיות שהוסברה בסעיף זה נקראת "**בדיקת זוגיות זוגית**" (**Even Parity**). לעיתים, בדיקת הזוגיות בודקת אם מספר הסיביות המורמות במסגרת הוא אי-זוגי. בדיקה כזו נקראת "**בדיקת זוגיות אי-זוגית**" (**Odd Parity**).



כתבו פונקציה המקבלת מסגרת נתונים בגודל בית ואשר הסיבית האחרונה בו היא סיבית זוגיות, ובודקת האם המסגרת תקינה.



אמינות בדיקת הזוגיות הזו היא **50%**, כיון שהיא תזהה כל שיבוש במספר אי-זוגי של סיביות בלבד.
נסו לחשוב על שיטה יותר אמינה,שתשתמש במספר קטן של סיביות לבדיקה ותזהה יותר מ-**50%** מהשגיאות.

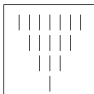
המשחק "נים" ("פירמידה הפוכה")

דוגמה נוספת לשימוש ב-XOR לבדיקת זוגיות של סיביות נוכל למצוא במשחקים שונים. במשחקים רבים בהם המשתמש משחק נגד המחשב, נדמה לנו שהמחשב "חושב" כמונו, כי הוא מצליח לשחק טוב ולעיתים אף טוב מאיתנו.

אמנם, לפעמים כך זה נראה, אך המחשב לא באמת חושב כמונו (לפחות נכון להיום). אם כך, איך בכל זאת המחשב מצליח לשחק כל כך טוב? הסיבה לכך נעוצה בעובדה שבמשחקים רבים מתחבא איזשהו טריק שבדרך כלל איננו שמים לב אליו. בעזרתו, ובעזרת חישובים שונים נוספים יכול המחשב לנצח אותנו במשחק. השחקן חושב על מהלכי המשחק ועל מה שעלול לקרות בעקבות כל צעד שייעשה, ואילו המחשב עושה חישוב מתמטי כלשהו ופועל לפי המסקנה הנובעת מכך.

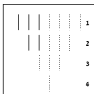
אחת הדוגמאות הפשוטות היא המשחק העתיק הנקרא "**נים**", אשר ידוע גם בשם "פירמידה הפוכה". אם השם לא מזכיר לכם דבר, אני חושב שלאחר שתקראו את התיאור שבהמשך, רובכם יזהו את המשחק מייד (בילדותי נהגתי לשחק במשחק זה בהפסקות הלימודים על הלוח, עם חברי לכיתה).

בתחילת המשחק מציירים על הדף או על הלוח, קווים אנכיים כמו בתרשים 2.1 (יש המשתמשים בגפרורים, במקום לצייר). המשחק מיועד לשני שחקנים, כל שחקן בתורו מוחק כמה קווים שהוא רוצה משורה אחת בלבד. המנצח הוא השחקן שמוחק את הקו האחרון שבלוח.


תרשים 2.1

אלה הם כללי המשחק. לפני שתמשיכו לקרוא, קראו לחבר ושחקו איתו פעמים אחדות, חשבו כיצד לפתח שיטות לנצח, או יותר נכון – כיצד לא להפסיד.
חשוב שתתנסו במשחק כדי שתוכלו להבינו.

אם הקשבתם לעצתי ושיחקתם במשחק, ודאי שמתם לב שבכל פעם שמגיע תורכם עליכם לנסות לחשוב מה מותר לעשות, כדי שבתור הבא השחקן השני לא יעשה משהו שיגרום לו לנצח (זה נשמע יותר מסורבל ממה שזה באמת).



תרשים 2.2

הביטו לדוגמה בתרשים 2.2 המציג תמונת מצב באמצע המשחק. נסו לחשוב איזה קו או קווים יש למחוק בתור זה כדי לנצח.

- אם תמחקו את כל הקווים שבשורה 2 או את כל הקווים שבשורה 1, השחקן השני ינצח בתור הבא.
- אם תמחקו קו אחד בשורה השנייה, השחקן השני ימחק כעת שני קווים משורה 1, ואז יוכל לנצח בסיבוב הבא.
- אם תמחקו שני קווים משורה 1, השחקן השני ימחק קו אחד משורה 2 ואז יוכל לנצח בסיבוב הבא.
- אם תמחקו קו אחד משורה 1, מובטח לכם שתנצחו! חשבו על כך.

ובכן, הבנו מה אנחנו צריכים לעשות כדי לשחק במשחק. אך איך נוכל לגרום למחשב לעשות מה שאנו רוצים שיעשה?

נבטא את כמות הקווים שבכל שורה בערכים בינאריים, ונבצע פעולת XOR בין ארבעת הערכים שיתקבלו. בתחילת המשחק הערכים ייראו כך:

```
      L1 = 7 = 111
      L2 = 5 = 101
      L3 = 3 = 011
      L4 = 1 = 001
L1^L2^L3^L4 = 000
```

כפי שלמדנו, ניתן לדמות שפעולת XOR לקחה בכל פעם רצף של ארבע סיביות מתאימות, בדקה אם מספר הסיביות המורמות בה הוא זוגי, החזירה אפס אם המספר זוגי, והחזירה אחד אם המספר אינו זוגי. התשובה הסופית של פעולת XOR בתחילת המשחק היא אפס.

הטריק במשחק זה: בתור שלכם עליכם למחוק מאחת ערמות קווים, כך שכשתעבירו את התור ליריב, תוצאת פעולת ה-XOR תהיה אפס. בדרך זו אין סיכוי שהיריב למשחק ינצח בתור הבא. אם תמשיכו לשמור שבכל פעם שתעבירו את התור פעולת XOR תחזיר אפס, אין דבר שהיריב יוכל לעשות כדי לנצח במשחק זה.

עכשיו אתם בוודאי מבולבלים. ראינו שהשחקן שמתחיל ראשון הוא המפסיד בוודאות. מכיון שהשחקנים האנושיים אינם מכירים את השיטה, אף אחד לא יטרח לחשב את פעולת ה-XOR. לכן, גם אם המחשב יתחיל ראשון, אין זה אומר שיפסיד, כי השחקן השני, האנושי, עלול לטעות. אולם, אם השחקן יתחיל ראשון והמחשב שני, המחשב תמיד ינצח! לכן, כדי לאפשר למשתמש לנצח, ניתן למחשב להתחיל במשחק.

נסו לבדוק טענות אלו וראו שהן נכונות.

עכשיו נותרה עוד בעיה קטנה. איך המחשב יידע איזה קו או קווים עליו למחוק, כדי להעביר את התור כך שפעולת XOR תחזיר אפס? הדבר מתבצע כך:

1. נמצא מהי השורה שנשארו בה מספר הקווים הרב ביותר.
2. נבדוק מה תהיה תוצאת פעולת XOR אם בשורה זו יהיה קו אחד פחות.
3. אם התוצאה תהיה אפס, נמחק בשורה זו קו אחד בלבד.
4. אחרת, נחזור על פעולות 2 ו-3 עד שנגלה כמה קווים יש למחוק מהשורה.



תמיד ישנה כמות כלשהי של קווים שאם נמחוק אותם מהשורה המלאה ביותר התוצאה של פעולת XOR תחזיר אפס, בתנאי שתוצאת XOR הנוכחית אינה אפס.

ובכן, מה נעשה אם תוצאת XOR בתחילת התור שלנו היא כבר אפס? במקרה כזה נמחק קו אחד בלבד מהשורה המלאה ביותר, ונקווה שבתור הבא, השחקן יטעה ולא יעביר לנו מצב שבו תוצאת XOR תהיה אפס.

נסכם את האלגוריתם שצריך לבצע המחשב:

1. אם לא נשארו קווים על הלוח, השחקן ניצח.
2. אחרת, מצא את השורה שנותר בה מספר הקווים הרב ביותר.
3. חשב את תוצאת פעולת XOR.
   - אם תוצאת XOR היא אפס, מחק קו אחד משורה זו.
   - אחרת, (השחקן טעה) מחק קווים משורה זו עד שתוצאת ה-XOR תהיה אפס. (אפשר לומר בוודאות שהמחשב ינצח במשחק זה).
4. אם לא נשארו קווים על הלוח, המחשב ניצח.

זה כל הרעיון שמסתתר מאחורי משחק זה. עכשיו נסו לכתוב את התוכנית שעושה זאת. קל מאוד לתכנת משחק זה, הוא יותיר רושם רב על כל מי שלא מכיר את הטריק, ולא יבין כיצד גרמתם למחשב לחשוב, כביכול.

לסיום, זכרו את שני השימושים שהודגמו בסעיף זה בפקודה XOR לבדיקת זוגיות בסיביות. בוודאי תוכלו לפתור בעיות נוספות על פי עיקרון זה.

פרק 3

שימוש בפסיקות ושילוב פקודות אסמבלי

בפרק זה נעסוק בפסיקות ובשילוב פקודות באסמבלי.

- ✔ **נלמד להפעיל פסיקות בשפת C.**
- ✔ **נלמד ליצור פסיקה משלנו שתחליף פסיקה קיימת.**
- ✔ **נלמד לשלב קטעי קוד בשפת אסמבלי לתוכנית בשפת C.**

3.1 פסיקות - הקדמה

פסיקה (**interrupt**) היא מנגנון המאפשר למעבד להפסיק את ביצוע התוכנית הנוכחית, לעבור זמנית לביצוע של תוכנית אחרת, ולחזור ולהמשיך את ביצוע התוכנית המקורית ממקום שהופסקה.

בניגוד לפרוצדורה, או פונקציה, המופעלות על פי פקודה פנימית בתוכנית, הפסיקה מופעלת הן בעקבות פקודה פנימית והן כתוצאה מאירוע חיצוני כמו למשל גלישה בפעולת חישוב, או פעולת חילוק שאינה מותרת (למשל, חילוק ב-0).

כדי שהמעבד יוכל לחדש כראוי את ביצוע התוכנית שהופסקה בתום ביצוע תוכנית השירות של הפסיקה (**ISR- Interrupt Service Routine**), עליו לעשות פעולות מכינות. לפני ביצוע הפסיקה עליו להכניס למחסנית את התוכן של שלושה אוגרים :

1. **אוגר הדגלים -Flags Register**. כך נוכל לשחזר את מצב הדגלים שהיה לפני ביצוע הפסיקה.
2. **אוגר מקטע הקוד Code Segment - CS**. כך נוכל לחזור למקטע הקוד של התוכנית ממקטע הקוד של הפסיקה **במידה והם לא זהים**.
3. **אוגר מצביע הפקודה Instruction Register - IP**. כך המעבד יידע באיזה פקודה הוא הפסיק את ביצוע התוכנית בזמן הפעלת הפסיקה.

בסיום ביצוע תוכנית השירות של הפסיקה, משוחזרים הערכים של שלושת האוגרים על פי הערכים שבמחסנית, וביצוע התוכנית המקורית ממשיך.

הערה

אם אינכם זוכרים מהו תפקידם של האוגרים השונים, קראו את נספח ו׳.

כתובות תוכניות השירות של הפסיקות השונות מאוחסנות ב**ווקטור הפסיקות**. המחשב יודע לטפל ב-256 פסיקות המזוהות על ידי מספר זיהוי הפסיקה.

לעיתים, תוכנית השירות של הפסיקה צריכה לקבל פרמטרים או להחזיר פרמטרים. לשם כך משתמשים באוגרים הכלליים : AX, BX, CX, DX.

באסמבלי מפעילים פסיקה באמצעות הפקודה int ואחריה מציינים את מספר הפסיקה המבוקשת. קביעת ערכי הפרמטרים שבאוגרים הכלליים נעשית על ידי הפקודה mov.

3.2 איגוד האוגרים המדומים ומבנה אוגרי המקטע

כדי להפעיל פסיקות בשפת C, יש להשתמש באיגוד של האוגרים המדומים -**union REGS**. הביטוי **union**, **איגוד**, נובע מכך שהמשתנים ״מולבשים״ זה על זה.

האיגוד נקרא **איגוד האוגרים המדומים** כיוון שאלה אינם האוגרים האמיתיים. לפני הפעלת הפסיקה, מועתקים ערכי האוגרים המדומים שבאיגוד (union) זה אל האוגרים האמיתיים, ובסיום ביצוע הפסיקה ערכי האוגרים האמיתיים (שברוב המקרים ערכם ישתנה כתוצאה מביצוע הפסיקה), מועתקים אל איגוד זה.

בשפת אסמבלי ניתן לפנות לאוגרים כאוגרים שלמים בגודל מילה (ax , bx , וכו׳) או כחצאי אוגרים בגודל בית (ax מתחלק ל- ah ול- al). כך, כל שינוי בחצי האוגר משנה את הסיביות המתאימות באוגר השלם. כדי לחקות צורת גישה זו, נשמרים האוגרים המדומים באיגוד (ראה סעיף 3.1). האיגוד נראה כך :

```
union REGS
{
  struct WORDREGS x;
  struct BYTEREGS h;
}

struct WORDREGS
{
  word ax;
  word bx;
  word cx;
  word dx;
  word si, di, cflag, flags;
};
```

```
struct BYTEREGS
{
  byte al, ah;
  byte bl, bh;
  byte cl, ch;
  byte dl, dh;
};
```

האיגוד מורכב משני מבנים ה״מולבשים״ זה על זה באותו שטח זיכרון. המבנה הראשון גדול יותר מהמבנה השני, ולכן גודל האיגוד נקבע על פיו. המבנה השני ״מתלבש״ על ארבע המילים הראשונות במבנה הראשון (תרשים 3.1).

.x	ax		bx		cx		dx		si	di	cflag	flags
.h	al	ah	bl	bh	cl	ch	dl	dh				

תרשים 3.1

הביטוי בדוגמה הבאה:

```
union REGS ireg;      // הגדרת איגוד מסוג איגוד האוגרים המדומים
ireg.x.ax = 0x1234;
```

לאחר ביצוע פקודה זו ireg.h.al שווה ל- 12H ו ireg.h.ah שווה ל- 34H.

יש פסיקות הזקוקות לפרמטרים שונים ב**אוגרי המקטע (Segment Registers)**. לשם כך הוגדר מבנה לשמירת הערכים של אוגרי המקטע. גם מבנה זה מכיל אוגרים מדומים, ולא את האוגרים האמיתיים. השימוש בו נעשה באופן דומה לשימוש באיגוד האוגרים המדומים.

המבנה נראה כך:

```
struct SREGS
{
  word es;  // Extra Segment
  word cs;  // Code Segment
  word ss;  // Stack Segment
  word ds;  // Data Segment
}
```

הערה

1. שימו לב שאת המילים "REGS" ו-"SREGS" יש לכתוב באותיות גדולות!
2. בראש התוכנית יש לכתוב: #include <dos.h>

3.3 הפעלת פסיקות בשפת C

כדי להפעיל פסיקות בשפת C נשתמש בפונקציות ספריה השייכות ליחידת הספריה dos.h.

int86

הפונקציה השימושית מכולן נקראת int86. היא תומכת בהפעלת כל הפסיקות הרגילות המתאימות למעבד בארכיטקטורת Intel 8086. הפונקציה מקבלת שלושה פרמטרים:

1. מספר הפסיקה שברצוננו להפעיל.
2. מצביע לאיגוד אוגרים מדומים שערכיו יעדכנו את האוגרים לפני הפעלת הפסיקה.
3. מצביע לאיגוד אוגרים מדומים שערכיו יעודכנו על פי האוגרים לאחר הפעלת הפסיקה.

ניתן להשתמש באותו איגוד לפעולות המפורטות הן בסעיף 2 והן בסעיף 3.

הפונקציה מחזירה את ערך אוגר AX כשהוא מוסב ל-integer, אך לרוב לא יהיה לנו צורך בערך המוחזר הזה כי הוא מוחזר גם בתוך איגוד האוגרים היוצאים (3).

הגדרת הפונקציה נראית כך:

```
int int86 (int intro, union REGS *inregs, union REGS *outregs);
```

הביטו בדוגמה הבאה וראו כמה זה פשוט. נניח שברצונכם להציג את סמן העכבר, ואתם יודעים שכדי לעשות זאת עליכם להפעיל את פסיקת העכבר שמספרה 33H עם פרמטר 1H באוגר AX. בעזרת int86 תוכלו לכתוב כך את הפונקציה להדלקת סמן העכבר:

```
void MouseOn (void)
{
  union REGS ireg;

  ireg.x.ax = 0x01;
  int86 (0x33,&ireg,&ireg);
}
```

נכון שאין זה מסובך?!

int86x

יש פסיקות הדורשות פרמטרים גם באוגרי המקטע. להפעלת פסיקות אלו נשתמש בפונקציה int86x. הפונקציה מקבלת את הפרמטרים הבאים:

1. מספר הפסיקה שברצוננו להפעיל.
2. מצביע לאיגוד אוגרים מדומים שערכיו יעדכנו את האוגרים לפני הפעלת הפסיקה.
3. מצביע לאיגוד אוגרים מדומים שערכיו יעודכנו על פי האוגרים לאחר הפעלת הפסיקה.
4. מצביע למבנה אוגרי המקטע.

הגדרת הפונקציה נראית כך:

```
int int86x (int into, union REGS *inregs, union REGS *outregs,
            struct SREGS * segregs);
```

אנו יכולים להבחין בכך שפונקציה זו זהה לפונקציה הקודמת, אלא שיש עוד תוספת של שליחת מבנה אוגרי המקטע. לרוב לא נצטרך להשתמש בפונקציה זו, אך דוגמה לשימוש בה תוכלו לראות בפרק 14, "כונן התקליטורים (CD-ROM)".

יש שתי פונקציות ספריה נוספות להפעלת פסיקות: **intdos** ו- **intdosx**. פונקציות אלו מקבילות לפונקציות int86 ו- int86x שהוסברו בסעיף זה, מלבד העובדה שהן תומכות בפסיקת DOS (21H) בלבד, ולכן אינן מקבלות את הפרמטר הראשון (מִסְפַּר הפסיקה). כיוון שניתן להפעיל את פסיקת DOS גם בעזרת שתי הפונקציות הרגילות (int86 ו- int86x) על ידי ציון מספר הפסיקה (21H), רצוי להשתמש בהן תמיד.

3.4 יצירת פסיקה חדשה ופלישה לפסיקה קיימת

לעיתים לא נסתפק בפסיקות הקיימות בספריה ונרצה ליצור בעצמנו תוכנית שירות לטיפול בפסיקה (ISR). לשם כך נכתוב פונקציה (שתשמש כתוכנית השירות החדשה), אך נכתוב לפני שמה את המילה **interrupt**. לדוגמה:

```
void interrupt MyIsr (void)
{
      קוד
}
```

הערה

לאלו מביניכם שמכירים את שפת אסמבלי, משמעות המילה השמורה interrupt היא, שבזמן תהליך ההיגור לאסמבלי נכתבת, בסוף הפונקציה המוגדרת בפסיקה, הפקודה iret במקום הפקודה ret.

את תוכנית הפסיקה שכתבנו עלינו לשייך לפסיקה כלשהי בווקטור הפסיקות. לשם כך נשתמש בפונקציה **setvect** המקבלת מספר פסיקה ואת שם תוכנית השירות ומעדכנת את וקטור הפסיקות. כך, התא של הפסיקה המתאימה בווקטור הפסיקות יקושר אל תוכנית השירות לפסיקה (ISR).

שימו לב שלא תדרסו פסיקה חשובה קיימת. לשם כך, בחרו בווקטור הפסיקות בתא שידוע לכם שאינו תפוס על ידי המערכת (שערכו מעל 100). בדוגמה שלהלן בחרנו בתא 103:

```
void SetIsr (void interrupt (*Isr) () )
{
   setvect(103,Isr);
}
```

לעיתים נרצה "לפלוש" לפסיקה קיימת וליצור תוכנית שירות (ISR) משלנו שתתבצע במקומה. אם נרצה שפקודות תוכנית השירות המקורית תתבצענה גם כן, נפעיל אותן מתוך תוכנית השירות החדשה שיצרנו.

הערה

המושג "פלישה" או "דריסה" (overriding) מכוון להחלפת תוכנית שירות, או פונקציה קיימת סטנדרטית בתוכנית אחרת שתישא את אותו שם ותכלול קוד חדש, או אף את הקוד הישן. כלומר, לא ניתן לשנות חלק של תוכנית השירות (ISR) המקורית! ניתן לכתוב תוכנית שירות חדשה שתחליף את הקיימת, ואם נרצה, נוכל להפעיל מתוכה את הקוד המקורי בשלמותו בלבד.

נעשה זאת כך:

1. ניצור תוכנית שירות (ISR) משלנו.
2. נשמור את כתובת תוכנית השירות המקורית (מצביע אליה) בתא ריק בווקטור הפסיקות.
3. נכניס לתא המקורי את כתובת תוכנית השירות שכתבנו (במקום כתובת תוכנית השירות המקורית).
4. בסוף ביצוע מערכת התוכנה שאנו כותבים, נשחזר את תוכנית הפסיקה המקורית על ידי העתקת הכתובת שלה אל התא המקורי (כדי שהתוכנות הבאות שנריץ במחשב יוכלו להשתמש בה).

כדי להעתיק כתובת (מצביע) של תוכנית שירות מתא לתא, ניעזר בפונקציה **getvect** המקבלת מספר פסיקה ומחזירה מצביע אל תוכנית השירות שלה.

נשמע מסורבל, אך אין זה כך, ואף פשוט ונוח יותר לכתוב זאת כך מאשר באסמבלי. לדוגמה, נכתוב תוכנית הפולשת לפסיקת המקלדת, ותפקידה לדמות קליטת סיסמה. בכל פעם שהמשתמש מקיש תו, תופיע על המסך כוכבית במקום התו שנקלט, אך התו עצמו ייכנס למחרוזת הסיסמה שבזיכרון.

כשלוחצים או משחררים לחיצה של מקש במקלדת, מופעלת פסיקת BIOS מספר 9. בעת קליטת תו, המקש גם נלחץ וגם משוחרר, ולכן פסיקה 9 מופעלת פעמיים. אם נפלוש לפסיקה 9 ונשים בה תוכנית שירות משלנו, שבנוסף להפעלת תוכנית השירות המקורית גם תדפיס כוכבית על המסך, יופיעו שתי כוכביות בכל קליטת תו. נוכל לפתור בעיה זאת אם נגדיר דגל גלובלי בשם Flag, לדוגמה, בכל הפעלת פסיקה נחליף את ערכו (מורם או לא מורם), ונדפיס כוכבית על המסך רק אם הוא מורם. התוכנית תיראה כך:

```
/*** KbIsr ***/

#include <stdio.h>
#include <conio.h>
#include <dos.h>

int Flag=0;
```

```
void interrupt MyIsr (void)
{
   union REGS ireg;
   int86(103,&ireg,&ireg);          // הפעלת הפסיקה המקורית ששמרנו בתא 103

   if (Flag)
      putch('*');
   Flag = !Flag;
}

void SetIsr (void interrupt (*Isr) () )
{
   setvect(103,getvect(9));        // העתקת הפסיקה המקורית לתא 103
   setvect(9,Isr);                 // העתקת הפסיקה שלנו לתא
}

void RestoreOldIsr (void)
{
   setvect(9,getvect(103));        // שחזור הפסיקה המקורית
}
char *GetPassword (void)
{
   char c=0,* PassW,i=0;

   PassW = (char *) malloc (20); // הקצאת מחרוזת של 20 תאים
   SetIsr(MyIsr);                // קביעת הפסיקה שלנו
   do
   {
      c = getch();
      PassW[i] = c;
      i++;
   }
   while (PassW[i-1]!=13);       // עד שלא נלחץ Enter

   PassW[i] = NULL;              // סוף מחרוזת
   RestoreOldIsr();              // שחזור הפסיקה המקורית
   return (PassW);
}

void main (void)
{
   puts(GetPassword());
}
```

כדי להימנע משמירת תוכנית השירות המקורית בתא שאינו ריק (כי גם אם התא אמור להיות ריק, ייתכן שתוכנה אחרת משתמשת בו), ניתן לשמור אותה במצביע לפסיקה. נעשה זאת כך:

- בראש התוכנית נגדיר מצביע לתוכנית השירות המקורית:

```
void interrupt (*OriginalISR)();
```

- את הפונקציה לאתחול תוכנית השירות שכתבנו נכתוב מחדש, כך:

```
void SetIsr (void interrupt (*Isr) () )
{
   OriginalISR = getvect(9);
    setvect(9,Isr);
}
```

- את הפונקציה לשחזור תוכנית השירות המקורית נכתוב מחדש, כך:

```
void RestoreOldIsr (void)
{
   setvect(9,OriginalISR));   // שחזור הפסיקה המקורית
}
```

3.5 שילוב פקודות אסמבלי בתוכנית

ניתן לשלב קטעי קוד הכתובים באסמבלי בתוכנית הכתובה בשפת C. מכנים זאת בשם **אסמבלי מוכלל - Inline Assembly**. למעשה, בספר זה לא היה צורך לשלב בתוכניות C גם פקודות אסמבלי מוכלל, גם בעת טיפול בפסיקות (שנעשה במקרים רבים בשפת סף), מכיון שהפונקציות שהכרנו בסעיפים הקודמים עושות את העבודה ביעילות ובקלות רבה.

אם בכל זאת תצטרכו לשלב פקודות אסמבלי בתוכנית C, תוכלו לעשות זאת כך: לפני כל פקודת אסמבלי כתבו את המילה **asm**. הפקודה תסתיים בסוף השורה, או בסימן נקודתיים (;).

```
asm mov ax,4
asm add ax,3; asm pop bx;
```

באסמבלי משמעות הסימן נקודתיים (;) הינה תחילת הערה. באסמבלי מוכלל משמעות הנקודתיים היא סוף פקודה (כמקובל בשפת C). הערה נכתבת כמקובל בשפת C, אחרי שני לוכסנים (//).

```
asm pop bx // הערה
```

ניתן לשלב מספר פקודות אסמבלי בבלוק פקודות אחד (סוגריים מסולסלים). פתיחת הבלוק **חייבת להיות** באותה שורה בה כתובה המילה asm.

לדוגמה,

```
asm {
  mov ax,4
  add ax,3; pop bx;
    }
```

פקודות אסמבלי המשולבות בשפת C יכולות להשתמש גם במשתנים שהוגדרו בשפת C, ולא רק באוגרים.

```
int x;
asm mov x,0xff
```

כמו בשפת אסמבלי, גם באסמבלי מוכלל **לא ניתן** לכתוב פקודה המשלבת שני משתנים.

```
asm add x,y // המהדר יודיע על שגיאה
```

נתגבר על בעיה זו כמו שנהגנו בתכנות אסמבלי – נעתיק את תוכן המשתנים לאוגרים ונחבר בין האוגרים:

```
asm{
  mov ah,x;
  mov al,y;
  add ah,al;
  }
```

התוכנית הבאה מדגימה את השימוש באסמבלי מוכלל כדי ליצור פונקציה המחברת בין שני מספרים ומודיעה על גלישה, כאשר פעולת החיבור גורמת להרמת דגל הגלישה (לתזכורת: דגל הגלישה מורם כאשר יש נשא בין הסיבית שלפני האחרונה, השמאלית ביותר, לבין הסיבית האחרונה, ואין נשא בין הסיבית האחרונה והחוצה).

```
/*** asm.c ***/

#include <stdio.h>
typedef unsigned char byte;

int Hiboor(byte a, byte b)
{
   int Overfllow=0 ;
   asm {
      mov al,a;
      mov ah,b;
      add al,ah;
      mov a,al;
      jno next;            // next אם אין גלישה קפוץ לתווית
      mov Overfllow,1;
      }
    next:
```

```
   if (Overfllow)
      printf("overfllow");
   return (a);
}

void main (void)
{
   byte x,y;
   x=60; y=2;
   printf("%d\n",Hiboor(x,y)); //לא תהיה גלישה
   x=64; y=64;
   printf("%d\n",Hiboor(x,y)); //יש גלישה
}
```

הערה

לא ניתן לכתוב תווית בתוך בלוק של פקודות אסמבלי מוכלל, ולכן כתבתי את התווית next מחוץ לבלוק.



פסיקת הגלישה היא מספר **4**. כתבו תוכנית בשפת C ללא שימוש באסמבלי מוכלל, כדי לבצע בדיוק את מה שמבצעת תוכנית הדוגמה הקודמת.
רמז: פלשו לפסיקת הגלישה והוסיפו בה את הדפסת ההודעה.

פרק 4

יצירת פרויקט, חלוקה מודולרית, תיעוד וסגנון

בפרק זה:

- ✔ **נלמד לחלק את מערכת התוכנה ליחידות.**
- ✔ **נלמד להרכיב פרויקט מאוסף יחידות.**
- ✔ **נלמד לתעד את המערכת.**
- ✔ **נלמד מספר כללי עיצוב סטנדרטים וסגנון.**

4.1 חלוקה ליחידות (units)

יחידה (**unit**) במערכת תוכנה, היא מודול **עצמאי,** המבצע אוסף תפקידים מוגדר המורכב מפונקציות, משתנים, טיפוסים וקבועים בעלי מכנה משותף תפקודי כלשהו.

אם כך, כיצד נדע מהם היחידות שירכיבו את מערכת התוכנה שאנו בונים?

כשמתכננים מערכת תוכנה, יש להתחשב ב**עיקרון המודולריות** (**modularity principle**) האומר כי היחידות (המודולים) המרכיבות את המערכת, צריכות להיות נוחות לחיבור והחלפה. כל יחידה תטפל בנושא אחד בלבד ותהיה עצמאית ככל האפשר. מערכת הבנויה בצורה מודולרית מגדילה את הסיכוי לשימוש חוזר בקוד שנכתב, שכן ניתן לשלב יחידות שנכתבו על פי עיקרון זה במערכות תוכנה שונות.

עצמאות היחידה מתבטאת ב**אי תלותה** ביחידות האחרות המרכיבות את המערכת. כאשר מפרקים את המערכת ליחידות, שואפים שהתלות בין היחידות השונות המרכיבות אותה תהיה קטנה ככל האפשר. באופן זה, גדל הסיכוי ששינויים עתידיים שיתבצעו במערכת יהיו מקומיים ופשוטים, וכך נִימָנַע מתופעת המפולת (שינוי קטן שגורר אחריו שינויים רבים לכל אורך מערכת התוכנה).

דוגמאות ליחידות:

- יחידה לטיפול בעכבר (מכילה את כל הפונקציות לתפעול העכבר).
- יחידת פלטים (מכילה את כל הפונקציות הקשורות להצגת הפלטים של המערכת על המסך: ציורים, חלונות, מסגרות וכו').
- יחידה להצגת תמונות BMP (מכילה את כל הפונקציות הנחוצות להצגת תמונה מסוג BMP על המסך).

כל יחידה מורכבת משני חלקים: חלק אחד הוא ה**מימוש** (**implementation**) והחלק השני הוא ה**ממשק** (**interface**). המימוש מכיל את הפונקציות המרכיבות את היחידה, ואילו הממשק מכיל את הגדרת הפונקציות והתיעוד בלבד.

הסיבה לחלוקה זו בין המימוש לבין הממשק נעוצה ב**עיקרון הסתרת המידע** (**information hiding principle**), הקובע כי ממשק היחידה צריך להכיל את המידע המינימלי הדרוש להפעלתה, בעוד שכל שאר מרכיבי היחידה מופיעים בחלק המימוש ומוסתרים מהמשתמש.

לרוב מערכת תוכנה מפותחת על ידי מספר מתכנתים, שכל אחד מהם אחראי על פיתוח יחידה מסוימת שצריכה לבצע פעולות מוגדרות. עיקרון הסתרת המידע מאפשר למתכנתים השונים העובדים על פיתוח המערכת להשתמש ביחידות התוכנה שכתבו חבריהם לעבודה, מבלי שיצטרכו ללמוד ולהבין כיצד מומשו הפונקציות המרכיבות אותן.

בשפת C, מימוש היחידה נשמר בקובץ שהסיומת שלו ".c", וממשק היחידה נשמר בקובץ בעל שם זהה ובעל סיומת ".h" (header).

את ממשק היחידה ניצור על ידי העתקת החלקים הבאים מקובץ המימוש:

- הגדרות מאקרו (#).
- הגדרות מבנים, איגודים וטיפוסים חדשים (typedef).
- הגדרת משתנים גלובליים (ללא אתחולם).
- הצהרות הפונקציות (prototypes) ותיעודן.

בכל פרויקט צריכה להיות **יחידה ראשית**, שהיא היחידה שמופיעה בה פונקציה בשם main. יחידה זו מאחדת את כלל היחידות המרכיבות את הפרויקט. ליחידה הראשית אין צורך ליצור קובץ ממשק.

כדי שנוכל להשתמש בפונקציות של יחידה אחת ביחידה אחרת, עלינו לזמן את היחידה בעזרת הפקודה **include** בדיוק כפי שאנו מזמנים את יחידות הספריה הסטנדרטיות (stdio.h, conio.h, וכו').

אם נרצה להשתמש במשתנה גלובלי שהוגדר ביחידה אחרת, עלינו להכריז עליו בראש התוכנית כמשתנה חיצוני, **extern**.

הערה

אם כותבים את שם היחידה בין סוגריים משולשים (< >), המהדר מחפש אותה בספריה include של המהדר. אם נכתוב את שם היחידה בין גרשיים (" "), המהדר יחפש אותה בספריה הנוכחית שבה אנו מריצים את הפרויקט.

שימו לב! טעות נפוצה היא לכתוב את מערכת התוכנה בקובץ אחד וברגע שהיא מוכנה, לחלק אותה ליחידות ספריה. כמעט כל מי שעושה זאת נתקל בסופו של דבר בקשיים בזמן החלוקה ליחידות, ועליו לבצע שינויים רבים כדי שניתן יהיה לחלק את הקוד. לכן, חובה **לתכנן מראש אלו יחידות תהיינה בפרויקט, לכתוב כל פונקציה ישירות ביחידה המתאימה, ולראות את מערכת התוכנה שבונים כפרויקט, כבר מתחילת התכנות!**

סכנת ה-goto

<שם תווית> goto – פקודה זו מקבילה לפקודה jmp באסמבלי ומשמשת כמקפצה לתוויות שונות לאורך התוכנית (הנמצאות לפני, או אחרי, פקודת הקפיצה). קפיצה ממקום למקום בתוכנית סותרת את רעיון התכנות המבני ואת עקרונות החלוקה המודולרית ויוצרת **קוד ספגטי** (**Spaghetti Code**) מסורבל ובלתי קריא. אם נתכנן כראוי את הקוד, נוכל להימנע לחלוטין מהשימוש בפקודה זו. במערכת התוכנה שנכתוב **לא נשתמש בפקודה goto**. יתרה מכך, שימוש בפקודה זו עלול "לעלות" לנבחנים בנקודות יקרות בציון שהבוחן מעניק לעבודה.

4.2 הרכבת הפרויקט

אחרי שהבנו כיצד לחלק את הקוד ליחידות, נלמד כיצד נחבר את היחידות לפרויקט. הכנתי כדוגמה פרויקט המורכב משתי יחידות הנקראות unit1 ו- unit2 ויחידה ראשית הנקראת MainF. תוכן הקבצים נראה כך (הקבצים שמורים בתיקיה של פרק זה):

```
/*** Unit1.h ***/

#include <stdio.h>
int Sum (int a, int b);  // returns the sum of a and b
```

```
/*** Unit1.C ***/
#include <stdio.h>

int Sum (int a, int b)
// returns the sum of a and b
{
   return (a+b);
}
```

```
/*** Unit2.h ***/
#include <stdio.h>

void Print (int x); // Prints x on screen
```

```
/*** Unit2.C ***/
#include <stdio.h>

void Print (int x)
// Prints x on screen
{
   printf("%d",x);
}
```

```
/*** MainF.C ***/

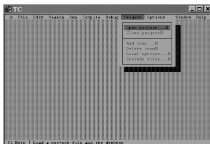
#include <stdio.h>
#include "unit1.h"
#include "unit2.h"

void main (void)
{
  Print(Sum(3,1));
}
```

שימו לב שלקובץ הראשי אין ממשק (h.). בקובץ הראשי השתמשנו בשתי הפונקציות הכתובות ביחידות unit1 ו- unit2 ולכן, בראש הקובץ קראנו לממשק של היחידות האלו. בדרך זו אנו קושרים את הממשק של היחידות אל הקובץ הראשי, אך כיצד מקושר המימוש (c.) של שתי היחידות?

כדי לקשר את המימוש של היחידות, עלינו לבצע את הפעולות הבאות:

1. נפתח פרויקט במהדר: תפריט Open Project < Project.



2. נבחר בשם לפרויקט עם סיומת prj.



3. בתפריט Project נבחר ב-Add item.

4. נבחר מתוך הרשימה את כל קבצי המימוש הקשורים לפרויקט, כולל הקובץ הראשי.

5. כדי להריץ את הפרויקט נלחץ Ctrl+F9.

הערה

אם היקף הפרויקט גדול, רצוי שלפני לחיצת Ctrl+F9 נפתח את כל הקבצים המרכיבים את הפרויקט ונלחץ בכל אחד מהם Alt+F9. באופן זה אנו מהדרים כל יחידה בנפרד לפני שאנו מקשרים ומריצים את הפרויקט כולו. כך אנו יכולים למנוע כשל בעת ההידור שיכול לנבוע מעומס שורות הקוד שהמהדר צריך לטפל בהן.

הערה

כדי למנוע הידור חוזר של קבצי Header, נהוג לכתוב בראש ה-Header הגדרת קדם מהדר הכוללת את שם הקובץ לדוגמה:

```
#define__MyUnit_H__
```

וכך בכל הקבצים, בהם נרצה לזמן את ה-Header, נוכל לבצע זימון מותנה:

```
#ifndef__MyUnit_H__
        #include <MyUnit.H>
#endif
```

4.3 מי אני ומה שמי?

בזמן כתיבת הקוד עליכם לזכור תמיד כי הקוד שאתם כותבים צריך להיות ברור וקריא ככל האפשר לכל מתכנת שיקרא אותו. אחד הדברים המשפיעים ביותר על בהירות הקוד (אף יותר מהתיעוד) הוא השמות שמעניקים למשתנים. אם תדעו לבחור בשמות משתנים טובים ומשמעותיים, הקוד "יסביר את עצמו" (Self documenting code) והמשתנים ישתלבו בקוד, כך שניתן יהיה לקרוא אותו בשטף ולהבינו ללא קושי.

מהו שם טוב?

- שם טוב הוא שם המרמז על התפקיד המלא של המשתנה ועל ההקשר שמשתמשים בו.
- אל תפחדו להשתמש בשמות ארוכים המורכבים מצירופי מילים. כדי שהשם יסביר את תפקידו המדויק של המשתנה, עליו להיות בדרך כלל בן שתיים עד שלוש מילים. לדוגמה, שימוש במשתנה בשם "mone", לא מסביר מה הוא מונה ולכן נוסיף לשם זה מילה נוספת שתתאר זאת.
- לעולם אל תשתמשו בשם משתנה אחד לשני יעודים שונים. לכל משתנה צריך להיות תפקיד **אחד** מוגדר מראש.

סגנון כתיבת השם

קיימים סגנונות כתיבה רבים ושונים לכתיבת שמות. כל חברת תוכנה קובעת תקן כתיבה משלה ומחייבת את העובדים וגם את הקבלנים שלה לפעול על פיהם. בהנחה שאינכם עובדים כרגע בחברה שיש לה תקן מחייב, או שאתם כותבים פרויקט בבית ספר, במכללה או באוניברסיטה, ולא חייבו אתכם לפעול על פי סטנדרטים מסוימים, אתם ראשים לבחור את הסגנון האהוב עליכם. אולם זכרו – ***עליכם להיות עקביים*** בכתיבת השמות בפרויקט!

יש שלושה סגנונות עיקריים לכתיבת שמות:

1. כל מילה בשם מתחילה באות גדולה ושאר האותיות הן רגילות. לדוגמה: NumOfStudents.

2. כל האותיות בשם יהיו רגילות והפרדה בין המילים בשם, מתבצעת על ידי קו תחתון (_) הצמוד למילים. לדוגמה : num_of_students.

3. בתחילת השם מופיעה תחילית קצרה באותיות רגילות המרמזת על טיפוס המשתנה, ואחריה כל מילה בשם מתחילה באות גדולה. לדוגמה : aStudents כדי לציין מערך תלמידים (a-array).

משתנים גלובליים ולוקליים: השמרנים מבדילים בין משתנים גלובליים לבין משתנים לוקליים או על ידי בחירת סגנון כתיבה שונה לכל אחד מהם, או על ידי הוספת סימן זיהוי כמו g קטנה לפני שם משתנה גלובלי.

קבועים: נהוג לכתוב קבועים באותיות גדולות בלבד. בין המילים המרכיבות את הקבוע כותבים קו תחתון (_). לדוגמה : #define VEC_SIZE.

כדי להמחיש את חשיבות בחירת שמות המשתנים, הביטו בפונקציה הבאה ונסו להבין מה היא מבצעת :

```
int func1 (void)
{
  int tmp, x  ;
  while(x<c)
  {
    if ( s[x].m <= num)
    tmp ++;
    x++;
  }
  return (tmp);
}
```

אני מניח שחוץ ממי שכתב אותה, אף לא אחד יידע לנחש מה תפקידה של הפונקציה. עכשיו ראו עד כמה ברורה הפונקציה כאשר החלפתי את שמות המשתנים לשמות משמעותיים :

```
int CountFailures (void)
{
  int NumOfFailures, MisparTalmid ;
  while( MispatTalmid <= CLASS_SIZE )
  {
    if ( Talmid[MisparTalmid].Mark < PASS_MARK )
      NumOfFailures ++;
    MisparTalmid ++;
  }
  return (NumOfFailures);
}
```

עכשיו הפונקציה הרבה יותר ברורה וקריאה, אבל יש בה עדיין דבר אחד שלא נהוג לעשות. נסו לחשוב מה לא בסדר בשמות המשתנים.

אני בטוח שרובכם לא הבחין בזאת, אך חלק משמות המשתנים שבפונקציה כתובים באנגלית וחלק כתובים בעברית (באותיות לועזיות). תופעה זו נפוצה מאוד אצל מתכנתים וגם אצל תלמידים מתוך עצלות להיות עקביים. אם תראו את הפונקציה הזו למתכנת מנוסה, הוא יבחין בזאת מייד. לכן, השתדלו להיות עקביים! או אנגלית או עברית, ורצוי **אנגלית**. עם זאת, לעיתים אנו מעדיפים עברית בתעתיק לטיני, כדי שתהיה תאימות בין שמות המשתנים לבין כותרות דוח, או שמות שדות בטופס, למשל. במקרים אלה מותר לחרוג.

4.4 תיעוד

בנוסף לבחירה נכונה של שמות משתנים, דבר נוסף המשפיע על בהירות הקוד הוא התיעוד.

שימו לב: בהמשך תמצאו הסבר כיצד כותבים תוכנית לפי הכללים שנלמדים בבתי הספר. מבנה תוכנית זה חשוב כאשר מגישים עבודה. מבנה התוכנית כפי שנלמד בהמשך (הרווחים בין השורות, המסגרות וכו') אינו חשוב כלל להרצת התוכנית ושלמותה. מהפרק הבא אנו נכתוב את התוכניות כפי שנהוג לכתוב בספרות המקצועית ולא לפי כללים אלה. (הערת ההוצאה)

תיעוד מובנה בתוך הקוד

מטרת התיעוד המובנה בתוך הקוד היא להסביר קטעי קוד **שאינם** מובנים מאליהם. **הערות מיותרות וסתמיות עלולות להזיק יותר מחוסר תיעוד**, מכיון שכאשר אין תיעוד, המתכנת צריך להבין מתוך הקוד מה קורה בתוכנית. תיעוד שגוי או מיותר עלול להטעות אותו ולגרום להנחות שגויות, או לבזבוז זמן בניסיונות להבין את מטרת ההערה (אם כתבו אותה אולי מסתתר בקוד משהו שלא שמנו לב אליו...). לכן, נתעד רק קטעי קוד מסובכים, פקודות לא שגרתיות וחלקים לוגיים מרכזיים.

הערה צריכה להיות ברורה ותמציתית ככל האפשר. לפני הערה יופיעו שני קווים אלכסונים (//), הערה המתייחסת לפקודה מסוימת תיכתב לצד הפקודה, והערה המתייחסת לקטע לוגי (אוסף פקודות בעל תכלית משותפת) תיכתב בשורה שלפניו.

הערה

שימו לב! חשוב לתעד את הקוד במהלך כתיבתו. דחיית התיעוד יוצרת תיעוד לא נאמן ולא מדויק. אם תכתבו את ההערות לאחר שתסיימו לכתוב את הקוד, סביר שתשכחו למה התכוונתם כשכתבתם ביטויים מסוימים.

לצערי, אני בטוח שלמרות הערה זו, רבים ידחו את התיעוד עד למועד שיהיה פנאי לעשות זאת, או עד שמנהל הפרויקט או המורה ידרשו זאת... ראו הוזהרתם, בסופו של דבר תבינו שלא כך צריך לנהוג.

תיעוד פונקציות

מטרת תיעוד הפונקציות היא להסביר למתכנת שלא כתב אותה, אך צריך להשתמש בה, מה היא עושה וכיצד להשתמש בה.

תיעוד הפונקציה חייב להכיל את החלקים הבאים:

1. תיאור הפונקציה: מה היא עושה, לשם מה, וכיצד היא עושה זאת.
2. תיאור הפרמטרים שהפונקציה מקבלת.
3. תיאור הערך שהפונקציה מחזירה.

אם מערכת התוכנה נכתבת על ידי יותר ממתכנת אחד, נהוג שכל מתכנת ששינה משהו בפונקציה יוסיף שורה לתיעוד הפונקציה ובה יכתוב את שמו, את תאריך העדכון ותיאור השינויים שביצע.

נהוג להקיף את תיעוד הפונקציה במסגרת כלשהי. לדוגמה:

```
int ChackBaseAddress (short AddressToCheck);
/*-----------------------------------------------------------------------------|
|    פונקציה הבודקת האם כתובת מסוימת היא כתובת הבסיס של ה-DSP.                   |
|    פרמטרים מועברים :                                                           |
|        AddressToCheck - הכתובת לבדיקה.                                         |
|    ערך מוחזר :                                                                 |
|        "אמת" אם הכתובת שנבדקה היא אכן כתובת הבסיס של ה-DSP.                   |
|-----------------------------------------------------------------------------*/
```

הערה

הטרחה לכתוב את המסגרות השונות בדוגמה זו ובהמשך אינה עולה לפעמים בקנה אחד עם חיסכון בזמן ויעילות העבודה. מילוי המסך בשורות מסגרת ורווחים אינו תמיד לתועלת הקריאות של הקוד, אלא דווקא עלול לסרבל, כי צריך לחפש את הקוד בין שפע ה"קישוטים" שסביבו. עם זאת, במקומות עבודה שונים ובבתי ספר, נהוג לעשות זאת. בהמשך נציג את הנוהג המקובל בבתי ספר ובמכללות. עם זאת נאמר, שבכל מקרה חובה לעשות הפרדה עניינית וסבירה בין קטעי קוד, קבוצות הוראות וכד', גם אם אין מגישים את העבודה לבדיקת ראש צוות, מורה, או בוחן.

תיעוד קבצים

בראש כל קובץ קוד יש לכתוב מסגרת המכילה את הנתונים הבאים:

1. שם הקובץ.
2. תיאור הקובץ.
3. הערות מיוחדות לגבי הקובץ (אם יש).

לדוגמה:

```
/*************************************************************************/
/*                              W A V . H                                */
/*                          ~~~~~~~~~~~~~~~~~~                           */
/*                  ממשק ליחידה להשמעת קבצי קול מסוג WAV                 */
/*                                                                       */
/* הערות : 1) היחידה תבצע זיהוי אוטומטי לכרטיס הקול ותמצא את כתובתו,     */
/*            את ה-IRQ ואת ערוץ ה-DMA.                                   */
/*         2) היחידה תומכת בקבצי WAV בפורמט 8 סיביות Mono סטנדרטי.       */
/*         3) הטיפול בהשמעת קבצי הקול מתבצע באמצעות פסיקות, לכן ההשמעה   */
/*            לא תפריע למהלך התוכנית !                                   */
/*************************************************************************/
```

נהוג לכתוב בראש הקובץ גם מי כתב אותו ובאיזה תאריך הקובץ הושלם.

4.5 מראה כללי

הגורם השלישי המשפיע על בהירות הקוד הוא המראה של הקוד (הזחות, רווחים וכו'). טכניקת עבודה פשוטה תאפשר לנו לחלק את הקוד חלוקה לוגית. כך נוכל לצמצם את העומס על עיני הקורא, לעזור לו להתמקד בחלקים החשובים ולהשפיע בצורה ניכרת על יכולת הקריאה השוטפת של הקוד.

היכן להשאיר שורות רווח?

- שורת רווח בין שורות include של יחידות הספריה הסטנדרטיות לבין שורות include של היחידות שיצרנו בעצמנו.
- שורת רווח בין שורות include לבין הגדרות המאקרו.
- 2-3 שורות רווח בין פונקציה לפונקציה.
- 1-2 שורות רווח בין הגדרת המשתנים לבין החלק הביצועי של הפונקציה.
- שורת רווח לפני כל קטע לוגי (אוסף פקודות בעל ות תכלית משותפת) בתוכנית.

רווחים והזחות

- לפני ואחרי אופרטור יש להשאיר רווח (למעט מקרים מועטים בהם זה ממש מפריע).
- בכל שורה נכתוב פקודה אחרת בלבד (למעט הפקודה switch בה מותר לכתוב break בהמשך שורת הפקודה case).
- בתנאים וביטויים מורכבים, נקיף כל פעולה בסוגריים (גם אם אין צורך טכני לעשות זאת).

- יש להקפיד על הזחה של שלושה רווחים ימינה בכל פקודה התלויה בפקודה הנמצאת לפניה (לדוגמה, הפקודות הנלוות לפקודה case או if).

מסגרות הפרדה

כדי לחלק את הקובץ וליצור הפרדה בין ההצהרות, הפונקציות והפונקציה הראשית, נשלב בקוד את המסגרות הבאות:

```
//////////////////////////////////////////////////////////////////////////////
//                           ה צ ה ר ו ת                                    //
//////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////////
//                          פ ו נ ק צ י ו ת                                  //
//////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////////
//                               M A I N                                    //
//////////////////////////////////////////////////////////////////////////////
```



ניתן לעצב את המסגרות שהוצגו בפרק זה בשיטות שונות, אך חשוב לשמור על עקביות ואחידות לכל אורך הקוד. לדוגמה, אפשר לרווח ביו הלוכסנים (/ / / / / /) או לכתוב תווים + (++++++) וכד'. במקרים רבים משתמשים בקו מרוסק (מקווקו), או שאין כותבים את אחת משורות הלוכסנים.

דוגמה מסכמת

הביטו בדוגמה הבאה המסכמת את כל מה שלמדנו בפרק זה. שימו לב לשורות הרווח, לרווחים, להזחות ולתיעוד.

```
/*** נכתב על ידי משה יצחקי בתאריך 6/2000 ***/


/****************************************************************************/
/*                          V i e w  B M P . C                              */
/*                          ~~~~~~~~~~~~~~~~~~                              */
/*                  יחידה להצגת תמונות השמורות בפורמט BMP                   */
/*                                                                          */
/*  הערות : 1) היחידה תציג על המסך כל תמונת BMP השמורה כ-"BMP צבעים 256"   */
/*             בפורמט הסטנדרטי של Windows.                                  */
/*          2) יחידה תומכת בצבע שקוף (פיקסל שהוגדר כשקוף לא יישתל על המסך) */
/****************************************************************************/
```

```
////////////////////////////////////////////////////////////////////////////
//                            ה צ ה ר ו ת                                 //
////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <dos.h>
#include <mem.h>

#include "VGA256.h"

/*** הכרזה על משתנים חיצוניים הנחוצים מהיחידה - VGA256 ***/
extern byte *VGA ;              // הכרזה על מצביע המסך
extern byte Pall[256][3] ;      // הכרזת מטריצת לוח הצבעים

struct BmpRec                   // רשומת תמונת BMP
   {
      word width ;              // רוחב התמונה
      word height ;             // גובה התמונה
      byte *data ;              // נתוני התמונה
   } ;

////////////////////////////////////////////////////////////////////////////
//                          פ ו נ ק צ י ו ת                               //
////////////////////////////////////////////////////////////////////////////

int ReadBMP (struct BmpRec *bitmap , char *FileName)
/*-------------------------------------------------------------------------|
|                                 פונקציה הקוראת קובץ BMP לזיכרון.        |
|                                                  פרמטרים מועברים :      |
|                                           bitmap* - רשומת BMP.          |
|                                           FileName* - שם הקובץ.         |
|                                                        ערך מוחזר :      |
|                                  אמת אם הפתיחה התבצעה בהצלחה.           |
|-------------------------------------------------------------------------*/
{
   FILE *fp ;
   long i ;
   word num_colors;
```

```c
///// פתיחת קובץ ה-BMP
if ((fp = fopen(FileName,"rb")) == NULL)
{
   printf("פתיחת קובץ ה-BMP נכשלה");
   return (0) ;
}

///// בדיקה האם זהו קובץ BMP תקין
if (fgetc(fp)!='B' || fgetc(fp)!='M')
{
   fclose(fp);
   printf("הקובץ אינו קובץ BMP תקין");
   return(0);
}

///// קרא את הנתונים החשובים מהקובץ
fseek (fp,16*sizeof(byte),SEEK_CUR) ;          // קפוץ 16 בתים קדימה
fread(&bitmap->width, sizeof(word), 1, fp);    // קרא את רוחב התמונה
fseek (fp,2*sizeof(byte),SEEK_CUR) ;           // קפוץ 2 בתים קדימה
fread(&bitmap->height,sizeof(word), 1, fp);    // קרא את גובה התמונה
fseek (fp,22*sizeof(byte),SEEK_CUR) ;          // קפוץ 22 בתים קדימה
fread(&num_colors,sizeof(word), 1, fp);        // קרא את מספר הצבעים
fseek (fp,6*sizeof(byte),SEEK_CUR) ;           // קפוץ 6 בתים קדימה
```

פרק 5

מקלדות ומדפסות

בפרק זה נעסוק במקלדת ובמדפסת.

- ✔ **נלמד לקרוא לחיצות על מקשים מיוחדים (מקשי החיצים, F1, F2 וכו').**
- ✔ **נלמד לבדוק את מצב המקשים הדביקים (CapsLock, NumLock וכו').**
- ✔ **נלמד לעקוב אחרי לחיצה ושחרור של מספר מקשים במקביל.**
- ✔ **נלמד להדפיס את תוכן המסך במדפסת.**
- ✔ **נלמד להדפיס מסמכים במדפסת.**

5.1 ASCII וקודים מורחבים

ASCII - **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (תקן אמריקאי סטנדרטי לחילופי מידע) הוא קוד מוסכם, שמקורו במכשירי הטלגרף הישנים ותפקידו לסמל כל תו במספר. תקן ASCII המקורי הותאם לשבע סיביות והכיל 128 (2 בחזקת 7) תווים שונים שערכים היה בין אפס ל-127.

המחשב משתמש בבתים של שמונה סיביות (ש"קיבולתם" 256 ערכים שונים), ולכן הוחלט להרחיב את הקוד ולהוסיף לו את הערכים 128 עד 255 הנקראים Extended ASCII Code (אסקי מורחב). ערכים אלה כוללים בין השאר את תווי השפה המקומית (לדוגמה, בישראל ערכים 128 - 154, מייצגים את האותיות העבריות א'-ת') ועוד סימנים שונים המשמשים לציור מסגרות וקווים שונים.

בזמן לחיצה על מקש במקלדת (או על שילוב מקשים), מוכנס לחוצץ המקלדת ערך ASCII של המקש, או של המקשים שנלחצו. כדי לשלוף את הערך מתוך החוצץ, משתמשים בפונקציות קלט, כגון: ()getch.

לעיתים נרצה לדעת בתוכנית האם המשתמש לחץ על מקש כלשהו, ולא איכפת לנו/לו מהו המקש. אם נשתמש בפונקציה ()getch, ריצת התוכנית תעצור ותחכה שייקלט מקש כלשהו. לכן, אם נרצה שהתוכנית תמשיך לבצע משהו עד שיילחץ המקש, נצטרך למצוא דרך חלופית. הפתרון פשוט למדי: נבדוק האם חוצץ המקלדת ריק, או שיש בו ערך או ערכים כלשהם. לשם כך נשתמש בפונקציית הספריה **()kbhit**. פונקציה זו אינה שולפת ערך

מהחוצץ, אלא בודקת את תכולת החוצץ. היא מחזירה "אמת" אם יש בו ערכים, ו"שקר" אם הוא ריק. הביטו למשל בדוגמה זו:

```
while (!kbhit())
{
     כל עוד לא יילחץ מקש תוכן הלולאה יתבצע
}
getch();
```

בסוף הלולאה הפעלתי את הפונקציה ()getch, כיון שהפונקציה ()kbhit לא שלפה את ערך המקש שנלחץ. אם הייתי רוצה לדעת מהו ערך התו שנלחץ, הייתי פשוט מציב את הערך שמחזירה ()getch במשתנה מסוג char.

עד כאן הכל טוב ופשוט, הבעיה מתחילה כשנלחצים מקשים שלא הוגדר עבורם קוד ASCII. לדוגמה, מקשי החיצים או מקשי הפונקציה F1) F, F2 וכו'). עבור מקשים אלה הוגדרו control key codes - קודי מקשי בקרה. כאשר נלחץ מקש שאין לו ערך ASCII, נדחפים לחוצץ המקלדת שני ערכים בגודל בית (במקום ערך אחד בגודל בית המציין את קוד ה-ASCII), הערך הראשון הוא אפס והערך השני הוא ערך ה- control key code.

טבלאות קודי ASCII וקודי מקשי הבקרה (control key codes), מופיעות בנספח ב' בספר זה.

כדי לקלוט את ערכי מקשים אלה, נכתוב פונקציה בשם GetKey הקולטת תו בעזרת ()getch. אם ערך התו שנקלט הוא אפס (כלומר, נלחץ מקש בקרה שאין לו קוד ASCII), נפעיל שוב את הפונקציה ()getch ונחזיר את ערך קוד הבקרה (control key code) שלו. הפונקציה מקבלת כמצביע ערך שיהפוך ל"אמת" כשנלחץ מקש מורחב, ו"שקר" – כשנלחץ מקש רגיל.

הביטו בתוכנית הבאה המכילה פונקציה זו ומדגימה את השימוש בה:

```
/*** Key.c ***/

#include <stdio.h>
#include <conio.h>

char GetKey (int *ControlKeyCode)
{
   char c;
   c = getch();
   if (c==0)
   {
      *ControlKeyCode = 1;
      c = getch() ;
   }
   else
      *ControlKeyCode = 0;
   return (c);
}
```

```
void main (void)
{
   char ch;
   int CKC;

   while (ch!=27)
   {
      ch = GetKey(&CKC);
      if (CKC)
         putch('C');
      printf("%u,",ch);
   }
}
```

כל עוד לא נלחץ מקש Esc, התוכנית תדפיס על המסך את ערך התו שנלחץ. אם נלחץ מקש מורחב תופיע האות e לפני ערכו. השתמשתי ב- u% כדי להדפיס את ערך התו, ולא את התו עצמו (ראו תרשים 1.3 בפרק 1).

חזרו לקטע הקוד שנכתב בתחילת הסעיף (הלולאה המחכה ללחיצה על מקש כלשהו), נסו לחשוב איזו בעיה עלולה להתרחש אם תשתמשו בו, וכיצד ניתן לפתור אותה.

אם המשתמש ילחץ על מקש מורחב במקום על מקש רגיל, ייכנסו לחוצץ שני ערכים, ריצת הלולאה תיפסק והפונקציה ()getch תשלוף ערך אחד בלבד. אם תכתבו לולאה נוספת אחריה, שתתבצע גם היא עד ללחיצת מקש, הלולאה לא תתבצע כלל, כי הערך השני עדיין נמצא בחוצץ. כדי לפתור את הבעיה, נחליף את הפקודה ()getch בלולאה הבאה, השולפת ערכים מהחוצץ כל עוד הוא אינו ריק:

```
while (kbhit())
  getch();
```

5.2 בדיקת המקשים הדביקים

"מקשים דביקים", הוא כינוי למקשי מקלדת הפועלים כמתג בעל שני מצבים (ON/OFF): Num Lock ,Caps Lock ו- Scroll Lock. הכינוי "דביקים" נובע מכך שלחיצה עליהם מחליפה את מצבם מ-OFF ל-ON ולהיפך (אין צורך להמשיך וללחוץ עליהם כדי שערכם לא ישתנה). לחיצה על מקשים אלה **אינה** מכניסה ערכים לחוצץ.

בנוסף למקשים הדביקים יש מקשים נוספים שלחיצה עליהם אינה מכניסה ערך לחוצץ, לדוגמה, לחיצה על מקש Shift או על Ctrl. אם המשתמש לוחץ על צירוף מקשים כגון Shift+2, יוכנס לחוצץ ערכו של התו '@', אך אם הוא ילחץ מקש Shift בלבד, לא יוכנס לחוצץ ערך כלשהו.

למרות זאת, במשחקי מחשב רבים מקשים אלה משמשים כמקשי המשחק ותוכנות רבות מתייחסות למצב שלהם. אם כך, כיצד עושים זאת?

כדי לבדוק את מצב מקשים אלה, נשתמש בפסיקה 16H עם פרמטר 2 באוגר ah, נעשה זאת בעזרת הפונקציה int86, כפי שלמדנו בפרק 3. לאחר הפסיקה יכיל אוגר al ערך בגודל בית המציין את מצב המקשים, וכל סיבית מציינת את מצבו של אחד המקשים. אם המקש לחוץ או המקש הדביק מופעל (ON), ערך הסיבית המתאימה יהיה 'אחד' (1).

הסיבית	המקש
b0	Shift ימני
b1	Shift שמאלי
b2	Ctrl
b3	Alt
b4	Scroll Lock
b5	Num Lock
b6	Caps Lock
b7	Insert

כדי לבדוק מקש רצוי מסוים, נשתמש בשיטת המסכות, כפי שלמדנו בפרק 2 , "עבודה בסיביות".

התוכנית הבאה מדפיסה על המסך שלוש הודעות על מצבם של שלושת המקשים הדביקים. אם נלחץ על אחד מהמקשים הדביקים, ההודעה תשתנה בהתאם. התוכנית תסתיים בלחיצה על מקש כלשהו (כל מקש מלבד השמונה שהוזכרו בסעיף זה).

```
/*** Locks.c ***/

#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define R_SHIFT     0x01
#define L_SHIFT     0x02
#define CTRL        0x04
#define ALT         0x08
#define SCROLL_LOCK 0x10
#define NUM_LOCK    0x20
#define CAPS_LOCK   0x40
#define INSERT      0x80
```

```
int KbStatus (void)
{
   union REGS ireg;
   ireg.h.ah = 0x02;
   int86 (0x16,&ireg,&ireg);

   return (ireg.h.al);
}

void main (void)
{
   int Status;

   while (!kbhit())
   {
      Status = KbStatus();
      clrscr();

      printf ("Caps Lock is ");
      if (Status&CAPS_LOCK)
         printf("ON\n");
      else
         printf("OFF\n");

      printf ("Num Lock is ");
      if (Status&NUM_LOCK)
         printf("ON\n");
      else
         printf("OFF\n");

      printf ("Scroll Lock is ");
      if (Status&SCROLL_LOCK)
         printf("ON\n");
      else
         printf("OFF\n");

      delay (100);
     }
     while (kbhit())   getch();
}
```

5.3 שילובי מקשי החיצים

אם פעם שיחקתם במרוצי מכוניות, ודאי שמתם לב, שברוב המשחקים מסוג זה, "חץ למעלה" מסמל את הגז, "חץ למטה" מסמל נסיעה אחורה והחיצים לימין ולשמאל מסמלים פניות ימינה ושמאלה בהתאמה. מסיבה זו, כשאתם משחקים במשחק ונוסעים קדימה, עליכם ללחוץ ללא הפסקה על "חץ למעלה", ובכל פנייה ללחוץ על החיצים לימין ולשמאל (בלי לעזוב את "חץ למעלה", כי אז תאבדו מהירות).

בדוגמה זו יש חשיבות לשילובי המקשים הלחוצים ולשחרור מקשים מסוימים בזמן שמקשים אחרים לחוצים. לא ניתן לעקוב אחרי כל זה בעזרת הפונקציות שהכרנו בסעיף הראשון, כי הן אינן מתייחסות לשחרור הלחיצה ולשילובי לחיצות (למעט שילובים הכוללים Shift). לכן, הדרך בה נפתור את הבעיה תהיה על ידי פלישה לפסיקת המקלדת 9H. תחילה עלינו להכיר קבוצת קודים נוספת: קודי סריקה (**Key Scan Code**).

לכל מקש במקלדת קיים קוד זיהוי (**Key Code**) בגודל 7 סיביות. בכל פעם שלוחצים או משחררים מקש במקלדת, הדרייבר של המקלדת שולח לפורט (port) 60H קוד בגודל שמונה סיביות, אשר נקרא ***קוד סריקה*** (**Key Scan Code**). שבע הסיביות הראשונות של הקוד הן קוד הזיהוי של המקש שנלחץ או שוחרר. הסיבית השמינית מסמלת את סוג הפעולה שהתבצעה, היא מכילה 0 (אפס) כאשר המקש נלחץ, ו-1 – כאשר המקש שוחרר.

אחרי שדרייבר המקלדת דוחף לפורט 60H את קוד הסריקה של האירוע (הלחיצה או השחרור), הוא מפעיל את פסיקת המקלדת המבצעת פעולות אלו:

1. קוראת מפורט 60H את קוד הסריקה (כדי לדעת מהו המקש והאם הוא נלחץ או שוחרר).
2. אם קוד הסריקה מסמל **לחיצה** על מקש או לחיצה על צירוף מקשים המוגדרים ברשימת קודי ASCII או קודי מקשי הבקרה, הקוד המתאים ייכנס לחוצץ המקלדת.
3. אם נלחץ או שוחרר אחד משמונת המקשים המיוחדים (סעיף קודם), תוכנית הפסיקה מרימה או מורידה את הדגל המתאים.
4. תוכנית הפסיקה מודיעה למקלדת, שהלחיצה או השחרור זוהו בהצלחה על ידי שליחת הערך 20H לפורט 20H.

במשך הזמן התברר ששבע סיביות אינן מספיקות לכל המקשים שבמקלדת, לכן הגדירו ***קודי סריקה מורחבים*** (**Extended Key Scan Codes**). ברגע שנלחץ או משוחרר מקש שלא קיים עבורו קוד סריקה רגיל (כמו למשל המקשים PageUP ו- PageDown), דרייבר המקלדת דוחף לפורט 60H את הערך e0H (224 עשרוני) ומפעיל את פסיקת המקלדת. הפסיקה שולפת את הקוד ומבינה כי נלחץ (או שוחרר) מקש מורחב, ומאשרת את שליפת קוד הסריקה המורחב. ברגע שדרייבר המקלדת מקבל את האישור, הוא דוחף לפורט 60H את ערך הקוד המורחב ומפעיל שוב את הפסיקה שמטפלת בו.

סביר להניח שאוסף הקודים עלול כבר לבלבל, לכן נסכם מה שלמדנו.

רשימות הקודים מתחלקים לשתי קבוצות:

1. קודי המקשים שדרייבר המקלדת שולח לפורט 60H. כל קוד מייצג מקש במקלדת (לתו '1' ולתו '!' יש אותו קוד, כי שניהם שייכים לאותו מקש במקלדת). קבוצה זו מתחלקת לשתי רשימות:

   1.1. **Key Scan Codes** – קודי הסריקה של המקשים הבסיסיים.

   1.2. **Extended Key Scan Codes** – קודי הסריקה של המקשים המורחבים. לפני קוד מורחב נשלח לפורט 60H הערך 224.

2. קודי התווים הנדחפים לחוצץ המקלדת על ידי הפסיקה. הקוד מתייחס לתו עצמו, או למקש הבקרה, ולאו דווקא למקשים שיצרו אותו (לתו '1' ולתו '!' יש קוד שונה). קבוצה זו מתחלקת לשלוש רשימות:

   2.1. **ASCII Codes** – רשימת קודי ASCII של התווים הבסיסיים שערכם בין 0 ל-127.

   2.2. **Extended ASCII Codes** – הרשימה המורחבת של קודי ASCII המיועדת לערכים 128 עד 256 (הכוללים בין השאר את תווי השפה המקומית, עברית).

   2.3. **Control Key Codes** – רשימת הקודים של מקשי הבקרה (חיצים, 12 מקשי הפונקציות ועוד). לפני קודים אלה נדחף לחוצץ המקלדת הערך אפס.

הערה

את חמש הרשימות תוכלו למצוא בנספח ב' שבסוף הספר.

נכון, זה קצת מורכב, וחבל שלא תכננו מראש רשימה אחת אחידה, אבל כשמבינים את חלוקת הקודים בין הרשימות אפשר להתגבר על כך.

עכשיו, לאחר הבנת תפקיד פסיקת המקלדת, נוכל לחזור לבעיה שלנו ולפתור אותה על ידי כתיבת תוכנית שירות לפסיקה (ISR) משלנו, שתפלוש לתוכנית הפסיקה הקיימת ותדרוס אותה בשלמותה או בחלקה. לפני שנעשה זאת, כדאי שנעיין בסעיף 3.4 שבפרק 3, שיש בו דוגמה המציגה 'פלישה' אל פסיקת המקלדת. אם אינכם זוכרים אותה, רצוי שתקראו אותה כעת שוב. בדוגמה זו החלפנו את תוכנית השירות המקורית של המקלדת בתוכנית שירות משלנו, שהדפיסה על המסך כוכבית ואחר כך הפעילה את תוכנית השירות המקורית שהועתקה אליה. כדי שהכוכבית לא תודפס פעמיים בכל הקשה (גם בלחיצה וגם בעזיבה) נעזרנו בדגל.

כדי לפתור את הבעיה של שילוב לחיצות במקשי החיצים, נפלוש לפסיקה זו ונכתוב תוכנית שירות משלנו עבור הפסיקה שתבצע את הפעולות הבאות:

1. קראו את קוד הסריקה.
2. אם נלחץ אחד ממקשי החיצים – הרימו את הדגל המתאים.
3. אם שוחרר אחד ממקשי החיצים – הורידו את הדגל המתאים.
4. אם נלחץ או שוחרר מקש אחר (לא אחד מהחיצים) – הפעילו את תוכנית השירות המקורית (ולמעשה את הקוד המקורי).
5. אם לא נלחץ או שוחרר מקש אחר, אין טעם להפעיל את תוכנית השירות המקורית, כי כבר טיפלנו בלחיצה/שחרור ולכן נשלח לפורט 20H את הערך 20H.

ארבעת הדגלים יאוחסנו בארבע הסיביות הראשונות של משתנה גלובלי בשם ArrowsStatus כך:

הסיבית	המקש
b0	Up
b1	Down
b2	Right
b3	Left

בצורה זו, לחיצה או שחרור של אחד ממקשי החיצים **לא** תשנה את דגליהם של שאר מקשי החיצים, כך נדע בכל רגע איזה מקשי חיצים לחוצים.

אם תעיינו ברשימות שבנספח ב', תראו שמקשי החיצים שייכים לרשימת קודי הסריקה המורחבים (Extended Key Scan Codes). כפי שכבר למדנו, בכל פעם שנלחץ (או משוחרר) מקש מורחב, הפסיקה מופעלת פעמיים (בפעם הראשונה בשביל לקלוט את הערך 224 ובפעם השנייה בשביל לקלוט את הערך המורחב). לכן, נוכל להתעלם מהעובדה שמקשי החיצים שייכים לקודים המורחבים ופשוט לבדוק, האם הערך שבפורט 60H הוא אחד מהערכים הבאים:

הקוד	משמעות
72	לחיצה על "חץ למעלה"
200	שחרור "חץ למעלה"
80	לחיצה על "חץ למטה"
208	שחרור "חץ למטה"
77	לחיצה על "חץ ימינה"

הקוד	משמעות
205	שחרור "חץ ימינה"
75	לחיצה על "חץ שמאלה"
203	שחרור "חץ שמאלה"



מקשי החיצים הרגילים ומקשי החיצים במקלדת הנומרית (Key Pad), כשמקש NumLock כבוי, מחזירים אותם ערכי KeyScanCodes.

התוכנית הבאה מדגימה את הפלישה לפסיקה ומציגה על המסך את מצבם של ארבעת מקשי החיצים בכל רגע, עד שנלחץ מקש שאינו אחד מארבעת החיצים שימו לב: אם לוחיצים או משחררים מקש חץ, ה-ISR המקורי אינו מופעל, ערכו לא נכנס לחוצץ, ולכן לחיצה על אחד ממקשי החיצים לא תגרום ליציאה מהלולאה ורק תשנה את ההודעה המתאימה על המסך.

```
/*** Arrows.C ***/

#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define UP     0x01
#define DOWN   0x02
#define RIGHT  0x04
#define LEFT   0x08

typedef unsigned char byte;
byte ArrowsStatus = 0;              // סטטוס מקשי החיצים

void interrupt (*OriginalISR)();    // מצביע ל-ISR המקורי

void interrupt MyISR ()             // ה-ISR שלנו
{
   byte CallOriginalISR = 1;        // הפעל את ה-ISR המקורי בתום ה-ISR

   byte scanCode = inportb (0x60);  // קריאת ה-Key Scan Code

   if (scanCode == 72)              // נלחץ חץ למעלה
   {
      ArrowsStatus |= UP;
      CallOriginalISR = 0;          // אין צורך להפעיל את ה-ISR המקורי
   }
```

```
if (scanCode == 200)              // שוחרר חץ למעלה
{
   ArrowsStatus ^= (UP);
   CallOriginalISR = 0;           // אין צורך להפעיל את ה-ISR המקורי
}

if (scanCode == 80)               // נלחץ חץ למטה
{
   ArrowsStatus |= DOWN;
   CallOriginalISR = 0;           // אין צורך להפעיל את ה-ISR המקורי
}

if (scanCode == 208)              // שוחרר חץ למטה
{
   ArrowsStatus ^= (DOWN);
   CallOriginalISR = 0;           // אין צורך להפעיל את ה-ISR המקורי
}

if (scanCode == 77)               // נלחץ חץ ימינה
{
   ArrowsStatus |= RIGHT;
   CallOriginalISR = 0;           // אין צורך להפעיל את ה-ISR המקורי
}

if (scanCode == 205)              // שוחרר חץ ימינה
{
   ArrowsStatus ^= (RIGHT);
   CallOriginalISR = 0;           // אין צורך להפעיל את ה-ISR המקורי
}

if (scanCode == 75)               // נלחץ חץ שמאלה
{
   ArrowsStatus |= LEFT;
   CallOriginalISR = 0;           // אין צורך להפעיל את ה-ISR המקורי
}

if (scanCode == 203)              // שוחרר חץ שמאלה
{
   ArrowsStatus ^= LEFT;
   CallOriginalISR = 0;           // אין צורך להפעיל את ה-ISR המקורי
}
```

```
   if (CallOriginalISR)              // אם נלחץ מקש אחר
      OriginalISR ();                // הפעל את ה-ISR המקורי
   else
      outportb (0x20, 0x20);         // הכר בפסיקת החומרה
}

void InitISR (void interrupt(*ISR)())// אתחול ה-ISR שלנו
{
   OriginalISR = getvect (9);
   setvect (9,ISR);
}

void RestoreISR ()                   // שחזור ה-ISR המקורי
{
   setvect (9, OriginalISR);
}

void main (void)
{
   InitISR(MyISR);

   while (!kbhit())
   {
      clrscr();

      printf ("Up Key is ");
      if (ArrowsStatus&UP)
         printf("ON\n");
      else
         printf("OFF\n");

      printf ("Down Key is ");
      if (ArrowsStatus&DOWN)
         printf("ON\n");
      else
         printf("OFF\n");

      printf ("Right Key is ");
```

```
      if (ArrowsStatus&RIGHT)
         printf("ON\n");
      else
         printf("OFF\n");

      printf ("Left Key is ");
      if (ArrowsStatus&LEFT)
         printf("ON\n");
      else
         printf("OFF\n");
      delay(100);
   }
   while (kbhit())    getch();
   RestoreISR ();
}
```

לסיכום: בשלושת הסעיפים האחרונים פתרנו שלוש בעיות שונות הקשורות למקלדת. הבעיה הראשונה נפתרה בעזרת שימוש נכון בפונקציות הספריה, הבעיה השנייה נפתרה על ידי שימוש בפסיקת ה-BIOS, והבעיה השלישית נפתרה על ידי כתיבת פסיקה חדשה המחליפה את פסיקת המקלדת.

ככל שהגענו לבעיה קשה יותר, גילינו שעלינו לרדת לרמה יותר נמוכה (פונקציות > הפעלת פסיקה > יצירת פסיקה). זו הגישה שבה תיפגשו במהלך הלימוד בספר, וחשוב שתאמצו אותה: כשעומדת בפניכם בעיה, נסו לפתור אותה בעזרת פונקציות. אם אין פונקציה מתאימה, חפשו פסיקה שתעזור לכם. אם הפסיקה המתאימה אינה מספיק טובה כדי לפתור את הבעיה שלכם, אל תפחדו לפלוש אליה, לתקן ולשנות אותה. בצורה זו תוכלו לפתור כל בעיה שתרצו.

5.4 הדפסה

ודאי תהיתם כיצד ניתן להדפיס (במדפסת, ולא על המסך) מתוך שפת C. בסעיף זה נלמד שתי שיטות הדפסה ותראו עד כמה זה פשוט.

הדפסת המסך

לעיתים נרצה להדפיס את תוכן המסך הנוכחי. כדי לעשות זאת, עלינו להפעיל את הפסיקה Print Screen - 5H. נעשה זאת בעזרת ()int86, כפי שלמדנו בפרק 3:

```
void PrintScreen (void)
{
  union REGS ireg;
  int86 (0x05,&ireg,&ireg);
}
```

פשוט ושימושי.

הדפסה רגילה

אם נרצה להדפיס דברים אחרים, כמו למשל את תוכנו של קובץ טקסט כלשהו, נצטרך להשתמש בשיטה אחרת. נעשה זאת בעזרת קובץ הפלט stdprn.

בכל פעם שתוכנית מופעלת, מוגדרים באופן אוטומטי חמישה רצפים, streams (מצביעים לקבצי טקסט). אלה אינם קבצים אמיתיים, אך בתוכנית ניתן להתייחס אליהם כאילו היו קבצי טקסט רגילים, לכתוב או לקרוא מהם בעזרת פונקציות הקריאה/כתיבה הרגילות.

שם	תיאור
stdin	standard input device – חוצץ המקלדת
stdout	standard output device – הפלט למסך
stderr	standard error output device – רשימת שגיאות התוכנית
stdaux	standard auxiliary device – הפלט ליציאת עזר (COM1)
stdprn	standard printer – הפלט למדפסת (LPT1)

ניקח כדוגמה את stdout. בכל פעם שהתוכנית רוצה לכתוב משהו למסך, היא כותבת אותו ל"קובץ" stdout אשר נכתב אל המסך. כדוגמה, ניקח פקודה זו:

```
printf("מחרוזת");
```

היא שקולה לפקודה:

```
fprintf(stdout,"מחרוזת");
```

אם כך, כשנרצה לשלוח מחרוזת למדפסת, עלינו לכתוב:

```
fprintf(stdprn,"מחרוזת");
```

המחרוזת יכולה להכיל את כל תווי הבקרה הרגילים (תרשים 1.4 בפרק 1), מלבד מספר שינויים:

- הקבוע "\n" (מעבר שורה) בפלט המוצג על המסך גורם למעבר לתחילת השורה הבאה. במדפסת קבוע זה גורם למעבר שורה. כדי לעבור לתחילת השורה נשתמש בקבוע "\r".
- כדי לקדם את הדף המודפס כדי שניתן יהיה להוציא אותו מהמדפסת בתום ההדפסה (Form feed) נשתמש בקבוע "\f".
- יש מדפסות המחייבות שבסיום כתיבת המחרוזות תיכתב מחרוזת עם הקבוע "\f", כדי שהמדפסת תדע שכל ההוראות הסתיימו והיא יכולה להדפיס.
- אם ההדפסה תהיה ארוכה מדף אחד, המדפסת תשחרר את הדף בסיום ההדפסה ותמשיך להדפיס בדף חדש, באופן אוטומטי.
- אם המדפסת אינה תומכת ב**גופנים צרובים בעברית**, או בגופנים הנטענים אליה בדרך כלשהי, האותיות העבריות לא יודפסו כראוי. עיינו במפרט המדפסת שלכם כדי לדעת אם היא תומכת בהם או לא.

תוכנית דוגמה

התוכנית הבאה מקבלת כפרמטר שם קובץ ומדפיסה אותו (אם אינכם זוכרים כיצד כותבים תוכניות המקבלות פרמטרים, קראו את סעיף 1.3 בפרק 1).

```
/*** print.c ***/

#include <stdio.h>

void Print (FILE *fp)
{
   char st[81];

   fseek (fp,0,SEEK_SET);
   fprintf(stdprn,"\n\n\n\r");

   while (!feof(fp))
   {
      fgets(st,80,fp);
      fprintf(stdprn,st);
      fprintf(stdprn,"\n\r");
   }
   fprintf(stdprn,"\f");
}

void main (int argc, char *argv[])
{
   FILE *fp;

   if (argc == 1)
   {
      printf ("פרמטר הקובץ להדפסה חסר\n");
      exit(1);
   }
   if (argc > 2)
   {
      printf ("יש יותר מדי פרמטרים\n");
      exit(1);
   }
   if ( (fp = fopen (argv[1],"rb")) == NULL )
   {
      printf ("לא ניתן לפתוח את הקובץ");
      exit(1);
   }

   Print(fp);
}
```

פרק 6

קבצים וספריות

בפרק זה נעסוק **בקבצים** (**Files**) **ובספריות** (**Directories**), הנקראות במערכות Windows בשם **תיקיות** - **Folders**.

- ✔ **נלמד לזהות את מאפייני הקובץ.**
- ✔ **נלמד להציג רשימת קבצים על פי שאילתה כלשהי.**
- ✔ **נלמד לסייר בין הכוננים והספריות השונות.**
- ✔ **נלמד להעתיק, להעביר ולמחוק קבצים וספריות.**

פרק זה מיועד לכל מי שרוצה לכתוב פרויקט בסגנון Norton Commander או מערכת Windows בסיסית.

6.1 מבנה המידע של הקובץ ffblk

ביחידה הסטנדרטית **dir.h** מוגדר מבנה, אשר לתוכו מוכנס כל המידע הדרוש לנו אודות קובץ או ספריה מסוימים. המבנה נראה כך:

```
struct ffblk
{
  char ff_reserved[21];           // שמור על ידי דוס
  char ff_attrib;                 // תכונות הקובץ
  int  ff_ftime;                  // זמן יצירת הקובץ
  int  ff_fdate;                  // תאריך יצירת הקובץ
  long ff_fsize;                  // גודל הקובץ
  char ff_name[13];               // שם הקובץ
};
```

- השדה **ff_attrib** מכיל ערך המסמל את מאפייני הקובץ (**Attributes**) על פי הטבלה הבאה המוגדרת ב-dos.h:

קבוע	תיאור
FA_RDONLY	קריאה בלבד
FA_HIDDEN	קובץ נסתר
FA_SYSTEM	קובץ מערכת
FA_LABEL	תווית
FA_DIREC	ספריה
FA_ARCH	ארכיב

תרשים 6.1

- השדה **ff_ftime**, הוא משתנה בגודל 16 סיביות שבהן מקודד הזמן בו נוצר הקובץ. הזמן מקודד כך:

  b0 - b4 : ערך המייצג את השניות.

  b5 - b10 : ערך המייצג את הדקות.

  b11 - b15 : ערך המייצג את השעות.

כדי לפענח את הצופן נוכל לכתוב פונקציה, לדוגמה:

```
void DecodeTime (int TimeCode,int *H,int *M,int *S)
{
   *H = (TimeCode >> 11) & 0x1f;         // שעה
   *M = (TimeCode >> 5 ) & 0x3f;         // דקה
   *S = (TimeCode & 0x1f) *2;            // שנייה
}
```

אם הפונקציה אינה ברורה לכם, חזרו על פרק 2.

- השדה **ff_fdate** הוא משתנה בגודל 16 סיביות, בהם מקודד התאריך בו נוצר הקובץ. התאריך מקודד כך:

  b0 - b4 : ערך המייצג את היום.

  b5 - b8 : ערך המייצג את החודש.

  b9 - b15 : ערך המייצג את השנה מ-1980.

כדי לפרק את הצופן נוכל לכתוב פונקציה כגון הפונקציה הבאה:

```
void DecodeDate (int DateCode,int *DD,int *MM,int *YY)
{
   *DD = DateCode & 0x1f;                          // יום
   *MM = (DateCode >> 5) & 0x0f;                   // חודש
   *YY = (( (DateCode >> 9) & 0x7f ) + 80) %100;  // שנה
}
```

למשתנה YY הוספנו 80, כי הערך שבסיביות מייצג את השנה החל ב-1980, ולקחנו את השארית של החלוקה במאה, כדי לתמוך בשנים שאחרי שנת 2000 (אחרת, בשנת 2001 לדוגמה, התוצאה תהיה 80+21 = 101 במקום 01).

- השדה **ff_fsize** מכיל ערך המסמל את גודל הקובץ בבתים.
- השדה **ff_name** מכיל את שם הקובץ (כולל הסיומת).

הערה

מבנה זה מתאים גם לספריות ולא רק לקבצים. במקרה שמוכנסים אליו מאפייני ספריה: השדה ff_attrib מכיל את ערך הקבוע FA_DIREC, השדות ff_ftime ו ff_fdate מכילים את זמן ותאריך יצירת הספריה, השדה ff_fsize **אינו** מכיל את גודל הספריה, והשדה ff_name מכיל את שם הספריה (ללא נקודה וסיומת, כי אין לספריה סיומת).

6.2 נתיבים

נתיב (path) הוא ביטוי המתאר מיקום של קובץ או של ספריה מסוימת. לדוגמה:

```
c:\turboc\bin\tc.exe
```

הנתיב מורכב מהחלקים הבאים:

1. **שם הכונן - drive**. בדוגמה שלנו "**C:**" (כולל נקודתיים).
2. **מיקום הספריה - dir**. בדוגמה שלנו "**\turboc\bin**" (בהתחלה ובסוף חייב להיות לוכסן הפוך).
3. **שם הקובץ - name**. בדוגמה שלנו "**tc**".
4. **סיומת הקובץ - ext**. בדוגמה שלנו "**.exe**" (הנקודה משמאל **שייכת** לסיומת).

הערה

הנתיב לא חייב להכיל את כל ארבעת הרכיבים. אם הנתיב אינו כולל שם כונן, מערכת ההפעלה מניחה שמדובר בכונן הנוכחי. אם הנתיב אינו כולל את מיקום הספריה, מערכת ההפעלה מניחה שזו הספריה הנוכחית.

יש הגבלה לאורך המקסימלי בתווים של כל אחת מהמחרוזות הללו. האורך המקסימלי מוגדר בתוך הקבועים הבאים (המוגדרים ב-dir.h):

תיאור	קבוע	ערך
אורך מחרוזת הנתיב	**MAXPATH**	80
אורך מחרוזת הכונן	**MAXDRIVE**	3
אורך מחרוזת הספריה	**MAXDIR**	66
אורך מחרוזת השם	**MAXFILE**	9
אורך מחרוזת הסיומת	**MAXEXT**	5

תרשים 6.2

כדי לפרק נתיב למרכיביו השונים נשתמש בפונקציה **fnsplit** המוגדרת כך:

```
int fnsplit (const char *path, char *drive, char *dir, char
            *name, char *ext );
```

הפונקציה מקבלת את הפרמטרים האלה:

- path - הנתיב שאנו רוצים לפרק.
- drive - מחרוזת שלתוכה יוכנס שם הכונן.
- dir - מחרוזת שלתוכה יוכנס מיקום ושם הספריה.
- name - מחרוזת שלתוכה יוכנס שם הקובץ.
- ext - מחרוזת שלתוכה תוכנס סיומת הקובץ.

הפונקציה מחזירה משתנה מסוג integer המציין איזה מרכיבים נמצאים בנתיב. כדי לבדוק מאפיין כלשהו עלינו לבצע פעולת AND בין ערך זה לבין המסכה המתאימה. אם התוצאה היא אמת, המרכיב המתאים נמצא בנתיב. המסכות שמורות בקבועים הבאים:

מסכה שבעזרתה נדע אם:	קבוע
קיים כונן בנתיב	**DRIVE**
קיימת ספריה בנתיב	**DIRECTORY**
קיים שם קובץ בנתיב	**FILENAME**
קיימת סיומת בנתיב	**EXTENSION**
קיימים הסימנים "*" או "?" בנתיב	**WILDCARDS**

תרשים 6.3

אם נרצה להרכיב נתיב בעזרת המחרוזות של מרכיביו השונים, נשתמש בפונקציה **fnmerge** שפעולתה הפוכה לזו של הפונקציה fnsplit. הפונקציה מוגדרת כך:

```
void fnmerge (const char *path, char *drive, char *dir, char
            *name, char *ext );
```

הערה

חלק מהמחרוזות שמהן הפונקציה מרכיבה את הנתיב עשויות להיות ריקות.

6.3 כוננים

מציאת הכונן הנוכחי (התורן)

כדי לגלות מהו הכונן הנוכחי, נשתמש בפונקציה **getdisk** המוגדרת כך:

```
int getdisk (void);
```

הפונקציה מחזירה ערך המציין את מספר הכונן הנוכחי (A = 0, B = 1 ...), כדי להציג את אות הכונן, נוסיף לערך זה את ערך ה-ASCII של התו A.

מעבר לכונן אחר

כדי לעבור מהכונן הנוכחי לכונן אחר, נשתמש בפונקציה **setdisk** המוגדרת כך:

```
int setdisk (int drive);
```

הפונקציה מקבלת ערך המציין את מספר הכונן שאליו אנו רוצים לעבור (A = 0, B = 1 ...).

הפונקציה מחזירה ערך המציין את ערך הכונן האחרון במחשב.

6.4 ספריות

מעבר לספריה אחרת

הפקודה CD (change directory) של DOS מאפשרת לעבור מספריה לספריה. בסעיף זה נבנה תוכנית שתפעל כמו פקודה זו.

כדי לעבור מספריה לספריה, נשתמש בפונקציה **chdir** המוגדרת כך:

```
int chdir (const char *dir);
```

הפרמטר dir הוא מחרוזת הזהה למחרוזת dir שהכרנו בסעיף 6.2.

- כדי לעבור לספריה שנמצאת בתוך הספריה הנוכחית, המחרוזת צריכה להכיל את הנתיב שבין הספריה הנוכחית אל הספריה המבוקשת. לדוגמה, אם אנו נמצאים בספריית השורש ורוצים לעבור לספריה bin שבספריה turboc שנמצאת בספריית השורש, המחרוזת dir תיראה כך: "turboc\bin" (ניתן להוסיף לוכסן הפוך בסוף המחרוזת אך אין זו חובה).
- כדי לעבור לספריה הנמצאת רמה אחת מתחת לספריה הנוכחית, המחרוזת צריכה להכיל שתי נקודות: "..".
- כדי לחזור לספריית השורש, המחרוזת צריכה להכיל לוכסן הפוך בלבד: "\".
- כדי לעבור לספריה שאינה נמצאת בתוך הספריה הנוכחית, המחרוזת צריכה להכיל את נתיב הספריה המלא, כשהוא מתחיל בלוכסן הפוך, כלומר, נובע מספריית השורש. לדוגמה, אם נמצאים בספריה Windows ורוצים לעבור לספריה bin, המחרוזת צריכה להיות "\turboc\bin".

הערך המוחזר יהיה "0" אם המעבר התבצע בהצלחה, וערכו יהיה "-1" אם המעבר נכשל ונתיב הספריה המבוקש אינו קיים.

אם נרצה לחקות את הפקודה CD של DOS נוכל לכתוב תוכנית כזו:

```
/*** MyCD.C ***/

#include <dir.h>
#include <stdio.h>

void main (int argc, char *argv[])
{
   chdir(argv[1]);
}
```

שימו לב שהתוכנית מקבלת פרמטרים (כפי שלמדנו בפרק 1).

לאחר שניצור תוכנית ביצוע (EXE) נוכל להשתמש בה כמו שהיינו נוהגים בפקודה CD רגילה.

יצירת ספריה חדשה

כדי ליצור ספריה חדשה נשתמש בפונקציה **mkdir** שהגדרתה נראית כך:

```
int mkdir (const char *dir);
```

פונקציה זו זהה במבנה לפונקציה chdir שהוסברה בסעיף קודם.



כתבו תוכנית שתחקה את הפקודה MD (Make Directory) של DOS.

מחיקת ספריה

כדי למחוק ספריה נשתמש בפונקציה **rmdir** שהגדרתה נראית כך :

```
int mkdir (const char *dir);
```

פונקציה זו זהה במבנה לפונקציה chdir שהוסברה בסעיף קודם.

מגבלות:

1. הספריה צריכה להיות ריקה מקבצים.
2. ספריה זו אינה יכולה להיות הספריה הנוכחית.
3. ספריה זו אינה יכולה להיות ספריית השורש.



כתבו תוכנית שתחקה את הפקודה RD (Remove Directory) **של** DOS.

קליטת נתיב הספריה הנוכחי (התורן)

כדי לקלוט את הנתיב הנוכחי בכונן הנוכחי, נשתמש בפונקציה **getcwd** המוגדרת כך :

```
char *getcwd ( char *path, int pathlen );
```

הנתיב הנוכחי ייכנס למחרוזת path.

הפרמטר pathlen מציין את אורך המחרוזת path.

הערך המוחזר יהיה NULL במקרה והביצוע נכשל, או מצביע ל-path בהצלחה.

כדי לקלוט את הנתיב התורן בכונן שאינו הכונן התורן נשתמש בפונקציה **getcurdir** המוגדרת כך :

```
int getcurdir ( int drive, char *dir );
```

הפרמטר drive הוא ערך הכונן, שאת הנתיב התורן **בו** אנחנו מחפשים.

הפרמטר dir הוא מחרוזת הזהה למחרוזת dir שהכרנו בסעיף 6.2.

הערך המוחזר יהיה אפס בהצלחה, ו-'1-' בכישלון.

6.5 קבצים

בתחילת הפרק הכרנו את מבנה המידע על הקובץ ffblk. בסעיף זה נלמד למצוא קובץ או קבצים מסוימים בספריה הנוכחית ולקבל את תוכן ffblk שלהם.

החיפוש יתבצע על פי הנתיב שנגדיר. הנתיב יהיה בנוי כמו שלמדנו בסעיף 6.2. מותר שיהיו בו תווי ההכללה '*' או '?'.

כדי למצוא את הקובץ הראשון בספריה הנוכחית שמתאים לנתיב הרצוי, נשתמש בפונקציה **findfirst** המוגדרת כך:

```
int findfirst (const char path, struct ffblk *fb, int attr);
```

הפרמטר path יכלול את פירוט הנתיב המבוקש.

הפרמטר fb יכלול את כתובת מבנה המידע של הקובץ (הפונקציה תמלא בעצמה את הערכים בתוך המבנה).

הפרמטר attr יכלול את המאפיינים של הקובץ או של הקבצים שאנו מחפשים, על פי הטבלה שבתרשים 6.1. אם אין מאפיינים מיוחדים, נרשום "0".

הערך המוחזר יהיה אפס, אם נמצא קובץ התואם את הנתיב, ו-'1-' אם לא נמצא.

נניח לדוגמה, שברצוננו למצוא את הקובץ הראשון שמתחיל באות M, ובכלל זה קבצים מוסתרים וקבצי מערכת. הפקודה המתאימה לעשות זאת תיראה כך:

```
findfirst( "c:\windows\m*.*", &fb, FA_HIDDEN|FA_SYSTEM );
```

בחיפוש רגיל, קבצים מוסתרים וקבצי מערכת אינם מוצגים. על כן, בפרמטר המאפיינים כתבנו שהקובץ שאנו מחפשים עשוי להיות גם מוסתר או קובץ מערכת.

כדי למצוא את הקובץ הבא התואם את הנתיב, נשתמש בפונקציה findnext המוגדרת כך:

```
int findnext ( struct ffblk *fb );
```

הפונקציה תחזיר "0" אם נמצא קובץ נוסף התואם את הנתיב ו-'1-' אם לא.

אם כך, כדי למצוא את כל הקבצים התואמים נתיב מסוים, נבצע את הפעולות האלו:

1. נחפש את הקובץ הראשון התואם את הנתיב.
2. כל עוד נמצא קובץ התואם את הנתיב, נחפש את הקובץ הבא התואם את הנתיב.

6.6 חיקוי הפקודה DIR

רובנו מכירים את פקודת DOS להצגת רשימת קבצים, הפקודה DIR. למי ששכחו, ולאלה שדילגו על עידן מערכת ההפעלה DOS, מוקדשות מילים אלו: הפקודה DIR מאפשרת להציג על פי מסכה כלשהי, רשימה של קבצים וספריות הנמצאים בתוך ספריה מסוימת. לצד כל קובץ מופיעים גודלו והתאריך והשעה בהם נוצר. לדוגמה:

```
dir a*.c
```

פקודה זו מציגה על המסך את כל הקבצים שבספריה הנוכחית ששמם מתחיל באות a (התו * הוא תו הכללה, המציין מספר כלשהו של תווים כלשהם, לאחר האות a במקרה זה), והסיומת שלהם היא C.

בסעיף זה נלמד להכין פקודה כזו בעצמנו!

הערות:

- לפני הצגת רשימת הקבצים התואמים את הנתיב והמסכה, הפקודה DIR מציגה על המסך גם את רשימת הספריות הנמצאות בנתיב זה. כדי לחקות זאת, לפני הצגת רשימת הקבצים, הצגנו את רשימת הפונקציות על ידי כך שבפרמטר attrib הגדרנו ספריה. אך, אם נבצע חיפוש של ספריה על פי מסכה של קובץ (*.*), החיפוש ייכשל! כדי לפתור את הבעיה, השתמשנו בפקודה לפירוק הנתיב לרכיביו השונים. הרכבנו נתיב הזהה לנתיב המקורי בכל חלקיו מלבד מחרוזת שם הקובץ, שהוגדרה כ-"*" (כדי שיוצגו כל הספריות), ומחרוזת הסיומת שהוגדרה כמחרוזת ריקה.
- אם יוכנסו בהרצת התוכנית ארגומנטים מיותרים, היא תודיע על בעיה ותציג את המבנה החוקי של הפקודה.
- אם בהרצת התוכנית לא יוכנס ארגומנט המייצג נתיב, היא תתייחס לזה כאילו הוכנס הנתיב "*.*" ותציג את כל הקבצים שבספריה הנוכחית.

```
/*** MyDir.C ***/

#include <dir.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>

void DecodeDate (int DateCode,int *DD,int *MM,int *YY)
{
   *DD = DateCode & 0x1f;                          // יום
   *MM = (DateCode >> 5) & 0x0f;                   // חודש
   *YY = (( (DateCode >> 9) & 0x7f ) + 80) %100;   // שנה
}
```

```
void DecodeTime (int TimeCode,int *H,int *M,int *S)
{
   *H  = (TimeCode >> 11) & 0x1f;                  // שעה
   *M  = (TimeCode >> 5 ) & 0x3f;                  // דקה
   *S  = (TimeCode & 0x1f) *2;                     // שנייה
}

int main (int argc, char *argv[])
{
   struct ffblk FileInfoBlock;                     // בלוק מידע על קובץ
   int Done, i=0;
   int DD, MM, YY;
   int H, M, S;
   long DirSize=0;                                 // סה"כ גודל הקבצים
   char drive[MAXDRIVE],dir[MAXDIR],name[MAXFILE],ext[MAXEXT];
   char dirpath[MAXPATH];
   ///// אם יש יותר מדי ארגומנטים
   if (argc > 2)
   {
      printf("Usage : mydir [Path] \n");
      exit (0);
    }
   fnsplit (argv[1],drive,dir,name,ext);
   fnmerge (dirpath,drive,dir,"*","");
   ///// הדפסת המידע על הספריות
   Done = findfirst(dirpath,&FileInfoBlock,FA_DIREC);
   while (!Done)
   {
      DecodeDate(FileInfoBlock_fdate,&DD,&MM,&YY);
      DecodeTime(FileInfoBlock_ftime,&H,&M,&S);

      printf("%-13s   <DIR>",FileInfoBlock_name);
      printf("   %02d/%02d/%02d",DD, MM, YY);
      printf("   %02d:%02d:%02d\n", H, M, S);

      Done = findnext (&FileInfoBlock);
   }

   if (argc == 1)            // אם אין ארגומנט נוסף הצג את כל הקבצים
      Done = findfirst("*.*",&FileInfoBlock,FA_RDONLY|FA_HIDDEN|FA_SYSTEM);
   else
      Done = findfirst (argv[1], &FileInfoBlock,
                                       FA_RDONLY|FA_HIDDEN|FA_SYSTEM);
```

```
   while (!Done)
   {
      i++;                                    // ספירת הקבצים
      DirSize += FileInfoBlock_fsize;         // חישוב הגודל המצטבר
      DecodeDate(FileInfoBlock_fdate,&DD,&MM,&YY);
      DecodeTime(FileInfoBlock_ftime,&H,&M,&S);

      printf("%-13s %7ld",FileInfoBlock_name, FileInfoBlock_fsize);
      printf("   %02d/%02d/%02d",DD, MM, YY);
      printf("   %02d:%02d:%02d\n", H, M, S);

      Done = findnext (&FileInfoBlock);       // מציאת הקובץ הבא
   }
   printf ("     %d file(s)     %ld bytes\n",i,DirSize);
   return(0);
}
```

תוצאה לדוגמה:

```
C:\TURBOC>mydir bin\*.c
.              <DIR>    29/02/00   16:32:06
..             <DIR>    29/02/00   16:32:06
WAVPLAY.C        357    06/03/00   18:34:16
SBWAV.C        16162    02/03/00   09:25:30
WAV.C          20009    05/03/00   22:04:50
WAV8.C         20011    05/03/00   22:03:28
WAVSB.C        13048    02/03/00   19:49:14
TESTSB.C         846    05/03/00   22:14:40
BINARY.C         266    09/05/00   10:22:48
KBISR.C          656    11/05/00   09:08:18
ASM.C            472    11/05/00   10:54:22
MYDIR.C         2236    12/05/00   12:33:12
MYCD.C           122    12/05/00   13:47:28
DRIVE.C          163    12/05/00   15:35:16
      12 file(s)      74348 bytes

C:\TURBOC>
```

אם ברצונכם ליצור תוכנה כדוגמת Norton Commander, מומלץ שתבנו רשימה מקושרת של רשומות ffblk. בכל מעבר לספריה עליכם לאתחל את הרשימה במבנים של כל הקבצים שבספריה הנוכחית. בצורה זו תוכלו להציג על המסך את הקבצים כשהם ממוינים על פי שם, תאריך או סוג (סיומת). בנוסף עליכם להגדיר שני מצביעים שאחד מהם יצביע לרשומה

העליונה ביותר המוצגת על המסך ואחד – אל התחתונה. בכל פעם שהמשתמש יגלול את רשימת הקבצים, עליכם להזיז את שני המצביעים ולעדכן את התצוגה שבמסך.

דוגמה לשימוש ברשימה מקושרת, תוכלו למצוא בפרק 12 העוסק באנימציה.

6.7 העתקת קבצים

כדי להעתיק קובץ (copy), עליכם:

1. לפתוח את קובץ המקור לקריאה.
2. לפתוח את קובץ היעד לכתיבה.
3. לקרוא בית מקובץ המקור.
4. לכתוב בית לקובץ היעד.
5. כל עוד לא הגעתם לסוף קובץ המקור, עליכם לבצע סעיפים 3 עד 4 שוב ושוב ...
6. לסגור את שני הקבצים.

האלגוריתם הבא מדמה את הפקודה copy של DOS.

```
/*** MyCopy.c ***/

#include <stdio.h>
#include <conio.h>
#include <string.h>

void error (int errorcode)
{
   switch(errorcode)
   {
      case 1: printf("Source file is missing.\n");               break;
      case 2: printf("Target file is missing.\n");               break;
      case 3: printf("Too many parameters.\n");                  break;
      case 4: printf("Can't copy a file into itself.\n");        break;
      case 5: printf("Can't open source file for output.\n");    break;
      case 6: printf("Can't open target file for input.\n");     break;
   }
   fcloseall();
   exit(errorcode);
}
```

```
void main (int argc, char *argv[])
{
   FILE *source, *target;
   int i=0;
   char ch;
   if (argc == 1)                    // פרמטר קובץ המקור חסר
      error(1);
   if (argc == 2)                    // פרמטר קובץ היעד חסר
      error(2);
   if (argc>3)                       // יש יותר מדי פרמטרים
      error(3);
   if (!strcmp(argv[1],argv[2]))     // קובץ המקור וקובץ היעד זהים
      error(4);
   if ( (source = fopen (argv[1],"rb")) == NULL )
      error(5);
   if ( (target = fopen (argv[2],"wb")) == NULL )
      error(6);

   printf ("file is being copied");
   while (!feof(source))
   {
      ch=fgetc(source);
      fputc(ch,target);
   }
   fcloseall();
   printf ("\rfile copied successfully\n");
}
```



נסו להוסיף לתוכנית פס התקדמות (progress bar).
רמז: קפצו לסוף הקובץ ושמרו את המיקום. בתוך הלולאה בדקו את היחס שבין המיקום הנוכחי לבין סוף הקובץ, ומלאו את פס ההתקדמות בהתאם.



נסו לכתוב פונקציה, שתעתיק את כל הקבצים המתאימים שבספריה למסכה כלשהי.
רמז: היעזרו בתוכנית mydir.c. במקום להדפיס את המידע על הקובץ, שלחו את הנתיב שלו אל פונקציה שתבצע העתקה.



כתבו תוכנית שתעתיק ספריה שלמה (שעשויים להיות בה גם תת-ספריות).
רמז: עליכם לפתוח ספריה חדשה במקום היעד הרצוי, ואז להפעיל את הפונקציה להעתקת קבצים על פי המסכה המבוקשת. כאשר תגיעו אל ספריה, הפעילו שוב את הפונקציה החיצונית (בצורה רקורסיבית).

6.8 מחיקת קבצים

מחיקת קובץ (erase) מתבצעת בעזרת הפונקציה unlink המוגדרת כך :

```
int unlink (char *path);
```

הפונקציה מקבלת כפרמטר את נתיב הקובץ המיועד למחיקה, ומחזירה "0", אם המחיקה התבצעה בהצלחה ו-1- בעת כישלון.

התוכנית הבאה מחקה את הפקודה Del של DOS.

```
/*** MyDel.c ***/

#include <stdio.h>
#include <conio.h>
#include <string.h>

void error (int errorcode)
{
   switch(errorcode)
   {
      case 1: printf("No file specified.\n");       break;
      case 2: printf("Too many parameters.\n");     break;
      case 3: printf("Can't open file.\n");         break;
      case 4: printf("Can't erase file.\n");        break;
   }
   fcloseall();
   exit(errorcode);
}

void main (int argc, char *argv[])
{
   FILE *fp;
   long size;
   char ch;
```

```
  if (argc == 1)                    // לא צוין הקובץ למחיקה
     error(1);
  if (argc >2)                      // יש יותר מדי פרמטרים
     error(2);
  if ( (fp = fopen (argv[1],"r+b")) == NULL )
     error(3);
  fseek (fp,0,SEEK_END);
  size = ftell(fp);                 // גודל הקובץ
  fclose (fp);
  if (unlink(argv[1]))              // מחיקת הקובץ
     error(4);
  printf ("\rThe file has been erased successfully - %ld bytes\n",size);
}
```



נסו לכתוב תוכנית שתעביר (הפקודה move) קובץ מספריה לספריה.
רמז: ההעברה תתבצע על ידי העתקת הקובץ אל מקום היעד ומחיקת הקובץ המקורי. כלומר, לפנינו שתי פעולות.



נסו לכתוב תוכנית שתמחק ספריה שלמה (ייתכן שבתוכה יש עוד תת-ספריות).
רמז: עליכם למחוק את כל הקבצים שבתוך הספריה, לפני שתוכלו להסיר את הספריה. אם בעת מחיקת קבצים אתם מגיעים לספריה, עליכם להפעיל שוב את הפונקציה החיצונית (בצורה רקורסיבית).

פרק 7

מסכי טקסט

בפרק זה נעסוק במסך, כשהוא מוגדר במצב טקסט ("מוד טקסט").

- ✔ **נלמד להשתמש בצבעים.**
- ✔ **נלמד להעתיק קטעי טקסט.**
- ✔ **נלמד לקבוע את מיקום וצורת הסמן.**
- ✔ **נלמד כיצד לגלול את תוכן המסך (Scrolling).**
- ✔ **נלמד לכתוב ולקרוא ישירות מחוצץ המסך.**

7.1 צבעים

מוד טקסט (text mode) הוא המצב הסטנדרטי, או ברירת המחדל, לריצת התוכנית שלנו (בפרק 9 נכיר מצב פעולה נוסף). מוד זה מאפשר לכתוב על המסך תווי ASCII בלבד, ברזולוציה של 25X80 וב-16 צבעים שונים.

הקבוע	ערך	תיאור
BLACK	0	שחור
BLUE	1	כחול
GREEN	2	ירוק
CYAN	3	טורקיז
RED	4	אדום
MAGENTA	5	סגול
BROWN	6	חום
LIGHTGRAY	7	אפור בהיר
DARKGRAY	8	אפור כהה
LIGHTBLUE	9	תכלת
LIGHTGREEN	10	ירוק בהיר
LIGHTCYAN	11	טורקיז בהיר

תיאור	ערך	הקבוע
ורוד	12	LIGHTRED
סגול בהיר	13	LIGHTMAGENTA
צהוב	14	YELLOW
לבן	15	WHITH
הבהוב	128	BLINK

כדי לבחור את הצבע לכתיבה על המסך נשתמש בפונקציית הספריה **textcolor()**, ונשלח לה את אחד מהערכים (או הקבועים) שבטבלה. אם נרצה שהטקסט יהבהב, נוסיף 128 לערך המספרי של הצבע. לדוגמה, בחירת צבע צהוב מהבהב, תיראה כך:

```
textcolor(14+128);
```

או כך:

```
textcolor(YELLOW+BLINK);
```

כדי לבחור את צבע הרקע לטקסט נשתמש בפונקציית הספריה **textbackground()**, ונשלח לה את אחד משמונת הצבעים הראשונים שבטבלה (לא ניתן לבחור כרקע צבעים 8-15 וגם לא רקע מהבהב).

כדי לכתוב על המסך במאפייני הצבע והרקע שבחרנו, נשתמש בפונקציות הספריה **cprintf()** ו- **cputs()** המקבילות לפונקציות printf() ו- puts(), מלבד העובדה שהן תומכות בצבעים שנבחרו בעזרת textcolor() ו- textbackground.

7.2 מחיקות

- כדי למחוק את תוכן המסך נשתמש בפונקציית הספריה **clrscr()**.
- כדי למחוק את הטקסט שבהמשך השורה הנוכחית (ממיקום הסמן עד סוף השורה בה הסמן נמצא), נשתמש בפונקציית הספריה **clreol()**.



שתי פונקציות אלו, יצבעו את הרקע של האזור המיועד למחיקה על פי צבע הרקע האחרון, שנבחר בעזרת textbackground().

7.3 חלונות

ניתן להגדיר חלונות טקסט בעזרת פונקציית הספריה window(), המקבלת את מיקום הפינה השמאלית עליונה ומיקום הפינה הימנית תחתונה. לדוגמה:

```
window(LeftBorder,TopBorder,RightBorder,ButtonBorder);
```

ברגע שהגדרנו חלון טקסט, התוכנית תתייחס לאזור החלון כאזור המסך, ולגבולות החלון– כגבולות המסך. אם לדוגמה נפעיל את הפונקציה ()clrscr לאחר הגדרת החלון, לא יימחק תוכנו של כל המסך, אלא רק תוכנו של החלון שהגדרנו. גם אם נכתוב בעזרת אחת מפונקציות הפלט לאחר הגדרת החלון, הטקסט ייכתב בתחומי החלון בלבד וירידת שורה תגרום לירידת שורה בתחום החלון.

כדי לחזור לאזור המסך המקורי יהיה עלינו להגדיר חלון טקסט בגודל המסך:

```
window(1,1,80,25);
```

7.4 העתקה והזזה של מלבן טקסט

קיימות פונקציות ספריה להעתקה והזזה של מלבן טקסט. בעזרת הפונקציות ניתן לשמור בחוצץ את תוכנו של אזור מלבני במסך, או לטעון לאזור מלבני במסך את התוכן של חוצץ.

המונח "מלבן טקסט" ממחיש את העובדה שמדובר באזור מלבני (בהמשך נלמד כיצד לשמור תוכן של אזור שאינו מלבני), גם כדי ליצור הבחנה בינו לבין חלונות טקסט, כי העתקה והדבקה של תוכן אזור מלבני מסוים אינה מגדירה אותו כחלון טקסט.

פונקציית הספריה **()gettext** משמשת לשמירת תוכנו של מלבן טקסט בחוצץ זיכרון. הפונקציה מוגדרת כך:

```
int gettext(int left,int top,int right,int bottom,void *dest)
```

הפונקציה מקבלת את מיקום הפינה השמאלית עליונה, הפינה הימנית תחתונה, ומצביע לחוצץ שבו יישמר תוכן המלבן. הפונקציה מחזירה "אמת" בהצלחה ו"שקר" בכישלון.

פונקציית הספריה **()puttext** משמשת להדבקת תוכנו של מלבן טקסט השמור בחוצץ זיכרון. הפונקציה מוגדרת כך:

```
int puttext(int left,int top,int right,int bottom,void *src)
```

הפונקציה מקבלת את מיקום הפינה השמאלית עליונה, הפינה הימנית תחתונה, ומצביע לחוצץ שממנו נקרא תוכן המלבן. הפונקציה מחזירה "אמת" בהצלחה ו"שקר" בכישלון.



גודל חוצץ הזיכרון צריך להיות כפול משטח המלבן (כי בנוסף לטקסט נשמרים גם מאפייני הצבע של כל תו במלבן)

לדוגמה, הפקודות לשמירה ולשחזור של תוכן מסך הן:

```
char Screen [4000];           // 25*80*2
gettext(1,1,80,25,Screen);    // שמירת תוכן המסך
clrscr();                     // מחיקת תוכן המסך
puttext(1,1,80,25,Screen);    // שחזור תוכן המסך
```

אם אתם תוהים מדוע צריך לשמור ולשחזר את תוכן המסך, הנה מספר דוגמאות למקרים שהדבר נחוץ:

- נניח שבתוכנית יש תפריט בראש המסך (למשל, כמו התפריט ב-Word), וכשלוחצים על אחת מהמילים הכלולות בו, נפתחת רשימת הפעולות שניתן לבצע בעזרתו. ברגע שנפתחת הרשימה, היא מסתירה את תוכן המסך. לכן, כדי שניתן יהיה לשחזר את המסך, צריך לשמור אותו לפני פתיחת הרשימה, ובתום השימוש ברשימה צריך להחזיר את המסך השמור, וכך הרשימה תיסגר.
- דוגמה נוספת, "תיבת כן/לא" ("Yes/No box"). תיבה זו מופיעה במרכז המסך ברגע שהמשתמש בוחר בפעולה מסוימת, והיא מכילה את הלחצנים "כן" ו-"לא". נניח לדוגמה, שכאשר המשתמש בוחר לצאת מהתוכנית מופיעה תיבה, שעליו לענות בה אם הוא בטוח שברצונו לצאת מהתוכנה. אם המשתמש יתחרט ויבחר בלחצן "לא", צריך יהיה לשחזר את תוכן המסך המסתתר מאחורי התיבה. לכן, לפני ציור התיבה נשמור את האזור שמאחוריה בחוצץ, ונשחזר אותו בתום השימוש בה.

בנוסף לפעולות שמירה ושחזור של מלבן טקסט, דרושה פונקציה להזזת מלבן טקסט. פונקציה זו **movetext()** מוגדרת כך:

```
int movetext (int left, int top, int right, int bottom, int
                                  destleft, int desttop);
```

הפונקציה מקבלת את מיקום הפינה השמאלית עליונה, את מיקום הפינה הימנית תחתונה של מלבן המקור, וגם את מיקום הפינה השמאלית עליונה של מלבן היעד. הפונקציה מחזירה "אמת" בהצלחה, ו"שקר" בכישלון.

7.5 הסמן

הסמן הוא הסימן המהבהב המתקדם בהתאם למיקום הצגת התו הבא על המסך, מקום ההקלדה. כדי להזיז את הסמן למיקום מסוים במסך נשתמש בפונקציית הספריה **gotoxy()**, המקבלת מיקום (קואורדינטות) על המסך ומזיזה אליו את הסמן.
דוגמה לכתיבת כוכבית במרכז המסך:

```
gotoxy(40,13);
putch('*');
```

כדי לדעת מהו המיקום הנוכחי של הסמן, נשתמש בפונקציית הספריה **wherex()**, המחזירה את מיקומו האופקי של הסמן, ובפונקציה **wherey()**, המחזירה את מיקומו האנכי של הסמן. דוגמה להזזת הסמן תו אחד אחורה:

```
if (wherex()>1)
  gotoxy(wherex()-1,wherey());
else
  gotoxy(80,wherey()-1);
```

כל זה טוב ויפה, אבל מה נעשה אם נרצה להעלים את הסמן?

נניח לדוגמה שבונים תוכנית להצגת קבצי טקסט (viewer). אין טעם בכך שהסמן יהבהב למשתמש בשטח המסמך, כי הבהוב זה יהיה חסר משמעות עבורו, מכיון שהוא אינו יכול להקליד דבר.

לשם כך עלינו להיעזר בפסיקת המסך 10H עם פרמטר 1 באוגר ah. פונקציה זו של פסיקת המסך משמשת לקביעת גובהו של הסמן (הרוחב של הסמן אינו ניתן לשינוי). גובה הסמל מיוצג על ידי שני ערכים: מיקום הגבול התחתון (ביחס למיקום התו שהסמן נמצא בו), שנכניס לאוגר cl, ומיקום הגבול העליון שנכניס לאוגר ch.

מיקום הגבולות נע בין הערכים 0 ו-7. לדוגמה: הגבול התחתון של הסמן הגדול ביותר (גובהו כגובה תו) יהיה 0, וגבולו העליון יהיה 7.

כדי להעלים את הסמן לחלוטין נציב בשני האוגרים את הערך (-1).

נעשה זאת בעזרת הפונקציה ()int86, כפי שלמדנו בפרק 3.

דוגמה לפונקציה להעלמת הסמן:

```
void HideCursor(void)
{
   union REGS ireg;

   ireg.h.ah = 0x01;
   ireg.h.ch =-1;
   ireg.h.cl =-1;

   int86(0x10, &ireg, &ireg);

}
```

והפונקציה לשחזור הסמן לגודלו הרגיל תיראה כך:

```
void RestoreCursor(void)
{
   union REGS ireg;

   ireg.h.ah = 0x01;
   ireg.h.ch =6;
   ireg.h.cl =7;

   int86(0x10, &ireg, &ireg);
}
```

7.6 גלילה (Scrolling)

הבה נמשיך בדוגמה של הסעיף הקודם, העוסק בתוכנית להצגת קבצי טקסט. רצוי שיהיה בצידו של המסך פס גלילה, אשר בעזרתו המשתמש יוכל לגלול את תוכן המסמך.

כדי לבצע את פעולת הגלילה, נשתמש בפסיקת המסך 10H עם פרמטר 6 לגלילה מעלה, ופרמטר 7 לגלילה מטה. לפני הפעלת הפסיקה:

- אוגר ah צריך להכיל את כיוון הגלילה (6 - למעלה, 7 - למטה).
- אוגר al צריך להכיל את מספר השורות שברצוננו לגלול.
- אוגר cl צריך להכיל את מיקום הגבול השמאלי של שטח הגלילה.
- אוגר ch צריך להכיל את מיקום הגבול העליון של שטח הגלילה.
- אוגר dl צריך להכיל את מיקום הגבול הימני של שטח הגלילה.
- אוגר dh צריך להכיל את מיקום הגבול התחתון של שטח הגלילה.
- ואוגר bh צריך להכיל את מאפייני הצבע של שטח הגלילה.

הערה

המיקומים (הקואורדינטות), שהאוגרים מקבלים, צריכים להתחיל מאפס ולא מאחד (כמו שנהגנו עד כה). לכן נוריד 1 מערכי המיקום שמקבלת פונקציית הגלילה.

דוגמה לפונקציית גלילה:

```
void Scroll(int x1, int y1, int x2, int y2, int num,
            int where,int color)
{
   union REGS ireg;

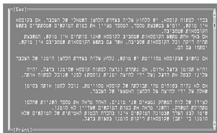
   ireg.h.ah = where;
   ireg.h.al = num;
   ireg.h.cl = x1-1 ;
   ireg.h.ch = y1-1 ;
   ireg.h.dl = x2-1 ;
   ireg.h.dh = y2-1 ;
   ireg.h.bh = color;

   int86(0x10, &ireg, &ireg);
}
```

7.7 תוכנית להצגת קבצי טקסט

הזכרנו פעמים רבות בפרק זה את הדוגמה של תוכנית להצגת קבצים (Viewer). כדאי שנתבונן בקוד של תוכנית כזו, המסכם את כל מה שלמדנו בפרק זה.

התוכנית מקבלת כפרמטר שם של קובץ טקסט (כפי שלמדנו בסעיף 1.3 בפרק הראשון), ומציגה אותו על המסך. בצידו הימני של המסך מופיע פס גלילה, שבעזרתו נוכל לגלול את תוכן המסמך המוצג. התוכנית גם מאפשרת להדפיס את תוכן המסמך בעזרת לחיצה על הלחצן p.



תרשים 7.1: תמונת המסך בזמן ריצת התוכנית

```
/*** Viewer.c ***/

#include <dos.h>
#include <stdio.h>

#define UP 6                        // גלילה כלפי מעלה
#define DOWN 7                      // גלילה כלפי מטה

void Scroll(int x1, int y1, int x2, int y2, int num, int where,int color)
{
   union REGS ireg;

   ireg.h.ah = where;
   ireg.h.al = num;
   ireg.h.cl = x1-1 ;
   ireg.h.ch = y1-1 ;
```

```
   ireg.h.dl = x2-1 ;
   ireg.h.dh = y2-1 ;
   ireg.h.bh = color;

   int86(0x10, &ireg, &ireg);
}

void HideCursor(void)
{
   union REGS ireg;

   ireg.h.ah = 0x01;
   ireg.h.ch =-1;
   ireg.h.cl =-1;

   int86(0x10, &ireg, &ireg);
}

void RestoreCursor(void)
{
   union REGS ireg;

   ireg.h.ah = 0x01;
   ireg.h.ch =6;
   ireg.h.cl =7;

   int86(0x10, &ireg, &ireg);
}

void Win (void)
{
   window(5,3,76,24);
   textbackground(1);
   textcolor(15);
   gotoxy(1,1);
}
```

```
void DrawTextViewer (void)
{
   int i;
   ///// ציור המסגרת
   textbackground(1);
   clrscr();
   textcolor(11);
   gotoxy(3,2);
   cprintf("§¯[    ]¯¯");
   gotoxy(3,24);
   cprintf("¦¯[      ]=");
   textcolor(10);
   gotoxy(6,2);
   cprintf("Esc");
   gotoxy(6,24);
   cprintf("Print");
   textcolor(11);
   gotoxy(78,2);
   cprintf("″ ");
   gotoxy(78,24);
   cprintf("• ");
   for (i=0;i<66;i++)
   {
      gotoxy(i+12,2);
      cprintf("¯");
      gotoxy(i+12,24);
      cprintf("¯");
   }
   for (i=0;i<21;i++)
   {
      gotoxy(3,i+3);
      cprintf("‶");
   }
   ///// ציור פס הגלילה
   textbackground(3);
   textcolor(1);
   gotoxy(78,3);
   cprintf("   ");
   gotoxy(78,23);
   cprintf("   ");
   Win();
}
```

```
void ScrollMark (long Pos,long ScrollSize)
/*--------------------------------------------------------------------------
                 פונקציה לעדכון פס הגלילה על פי המיקום הנוכחי בקובץ.
                                                   פרמטרים מועברים :
                 Pos - מיקום השורה הראשונה המוצגת על המסך.
                          ScrollSize  - גודל שטח הגלילה בקובץ.
                                                   ערך מוחזר : אין.
--------------------------------------------------------------------------*/
{
   int i;
   long MarkPos;
   ///// מחיקת הפס הקודם
   window(78,4,79,22);
   clrscr();
   textcolor(3);
   for (i=1;i<20;i++)
   {
      gotoxy(1,i);
      cprintf("Š");
   }
   ///// ציור הפס החדש
   MarkPos = Pos*18/ScrollSize;          // (כי פס הגלילה באורך 18 שורות)
   if (!MarkPos) MarkPos=1;
   gotoxy(1,MarkPos) ;
   textbackground(3);
   textcolor(1);
   cprintf("ÿ");
   Win();
}

void Down (long *FirstLine,long *LastLine,FILE **fp)
/*--------------------------------------------------------------------------
          פונקציה הקוראת את השורה הבאה מהקובץ ומעדכנת את חלון הטקסט.
                                                   פרמטרים מועברים :
  FirstLine  - מצביע העומד על סוף השורה בקובץ המוצגת בראש החלון.
  LastLine  - מצביע העומד על סוף השורה בקובץ המוצגת בסוף החלון.
                    fp  - מצביע למצביע הקובץ.
                                                   ערך מוחזר : אין.
--------------------------------------------------------------------------*/
{
   char st[76],c;
   fseek(*fp,*LastLine,SEEK_SET);         // קפוץ לסוף השורה האחרונה שהוצגה
   c=fgetc(*fp);                          // קרא תו מהקובץ
```

```
   if (!feof(*fp))                          // אם זהו לא סוף הקובץ
   {
      fseek(*fp,*LastLine,SEEK_SET);        // קפוץ לסוף השורה האחרונה שהוצגה
      fgets(st,80,*fp);                     // קרא שורה מהקובץ
      *LastLine = ftell(*fp);               // עדכן את מצביע השורה האחרונה
      Scroll(5,3,75,23,1,UP,1<<4);          // גלילת תוכן המסך כלפי מעלה
      gotoxy(1,21);                         // כתיבת השורה החדשה שקראת
      cprintf("%s",st);
      fseek(*fp,*FirstLine,SEEK_SET);       // קפוץ למצביע השורה הראשונה
      fgets(st,76,*fp);                     // קרא שורה
      *FirstLine = ftell(*fp);              // עדכן את המצביע
   }
}

void FindPrevLine (long *Pos,FILE **fp)
/*--------------------------------------------------------------------------
                         פונקציה המוצאת את סוף השורה הקודמת בקובץ.
                                                       פרמטרים מועברים :
  Pos  -  מצביע העומד על שורה מסוימת ומתעדכן במיקום השורה הקודמת.
   fp  -  מצביע למצביע הקובץ.
                                                            ערך מוחזר : אין.
--------------------------------------------------------------------------*/
{
   char c=0;
   long P;
   P = *Pos - 2*sizeof(char);
   ///// כל עוד לא נמצאה סוף שורה ולא הגענו לתחילת הקובץ, קפוץ תו אחד אחורה
   while (c!=10 && P>0)
   {
      P-=sizeof(char);
      fseek(*fp,P,SEEK   SET);
      c=fgetc(*fp);
   }
   if (P>0)
      *Pos = ftell(*fp);
   else if (P==0)
      *Pos = ftell(*fp)-sizeof(char);
}
```

```
void Up (long *FirstLine,long *LastLine,FILE **fp)
/*---------------------------------------------------------------------------
  פונקציה הקוראת את השורה שלפני השורה שבראש החלון ומעדכנת את תוכנו בהתאם.
  פרמטרים מועברים :
  FirstLine  -  מצביע העומד על סוף השורה בקובץ המוצגת בראש החלון.
  LastLine  -  מצביע העומד על סוף השורה בקובץ המוצגת בסוף החלון.
  fp  -  מצביע למצביע הקובץ.
  ערך מוחזר : אין.
--------------------------------------------------------------------------*/
{
   char st[76];
   long FL ;
   FL=*FirstLine;
   ///// אם השורה הראשונה המוצגת אינה הראשונה בקובץ הזז את המצביעים שורה אחורה
   FindPrevLine(&FL,fp);
   if (FL==0)
      return;
   else
      *FirstLine=FL;
   FindPrevLine(FirstLine,fp);
   FindPrevLine(LastLine,fp);
   ///// קרא שורה מהקובץ ועדכן את חלון הטקסט
   fseek(*fp,*FirstLine,SEEK   SET);
   fgets(st,76,*fp);
   *FirstLine = ftell(*fp);
   Scroll(5, 3, 75, 23, 1, DOWN,1<<4);
   gotoxy(1,1);
   cprintf("%s",st);
}

void Print (FILE **fp)
/*---------------------------------------------------------------------------
  פונקציה להדפסת תוכנו של קובץ טקסט.
  פרמטרים מועברים :
  ft  -  כתובת מצביע הקובץ.
  ערך מוחזר : אין.
--------------------------------------------------------------------------*/
{
   char st[81];
   fseek (*fp,0,SEEK   SET);
   fprintf(stdprn,"\n\n\n\n\n\n\r");
```

```
   while (!feof(*fp))
   {
      fgets(st,80,*fp);
      fprintf(stdprn,st);
      fprintf(stdprn,"\n\r");
   }
   fprintf(stdprn,"\f");
}

void Init (void)
/*------------------------------------------------------------------------
                          פונקציה לאתחול הדברים הדרושים להצגת קובץ העזרה.
                                                  פרמטרים מועברים : אין.
                                                        ערך מוחזר : אין.
------------------------------------------------------------------------*/
{
   HideCursor();
   DrawTextViewer();
   ScrollMark(1,18);
}

//////////////////////////////////////////////////////////////////////////
//                                M A I N                               //
//////////////////////////////////////////////////////////////////////////

void main (int argc, char *argv[])
{
   FILE *fp;
   char st[81],c;
   long FirstLine , LastLine ,ScrollSize;
   int i,Quit=0;
   if (argc == 1)
   {
      printf ("פרמטר הקובץ להדפסה חסר\n");
      exit(1);
   }
   if (argc > 2)
   {
      printf ("יש יותר מדי פרמטרים\n");
      exit(1);
   }
```

```
   if ( (fp = fopen (argv[1],"rb")) == NULL )
   {
      printf ("לא ניתן לפתוח את הקובץ");
      exit(1);
   }
   ///// הדפסת השורות הראשונות
   Init();
   fseek(fp,0,SEEK_END);
   ScrollSize = ftell(fp);
   fseek(fp,0,SEEK_SET);
   i=0;
   while (!feof(fp) && i++<21)
   {
      fgets(st,80,fp);
      cprintf("%s",st);
      gotoxy(1,wherey());
   }
   LastLine = ftell(fp);
   ScrollSize -= (LastLine);
   fseek(fp,0,SEEK_SET);
   fgets(st,80,fp);
   FirstLine=ftell(fp);
   while (!Quit)                  // כל עוד לא נלחץ לחצן היציאה
   {
      while ( !kbhit() );        // כל עוד לא נלחץ מקש
      c=getch();
      if (c=='2')                // גלילה מטה
      {
         Down(&FirstLine,&LastLine,&fp);
         ScrollMark(FirstLine,ScrollSize);
      }
      if (c=='8')                // גלילה מעלה
      {
         Up(&FirstLine,&LastLine,&fp);
         ScrollMark(FirstLine,ScrollSize);
      }
      if (c==27)                 // יציאה
         Quit =1;
      if (c=='p')
         Print(&fp);
   }
   ///// יציאה
   fclose(fp);
   RestoreCursor();
}
```

7.8 גישה ישירה לחוצץ המסך

אם ניסיתם בעבר לשתול תו בפינה הימנית-תחתונה של המסך, ודאי גיליתם שמתבצעת פעולת גלילה אוטומטית של המסך וכל השורות עולות מעלה. ובכן, לכל המתוסכלים שהתייאשו מניסיונותיהם לשתול תו בפינה זו, הרי לכם דרך לעשות זאת.

עד כה, בכל פעם שרצינו לכתוב למסך, השתמשנו בפונקציות אשר השתמשו בפסיקות לשתילת התווים על המסך. דרך מהירה יותר לעשות זאת, היא לפנות לחוצץ המסך ולכתוב בו ישירות את התווים הרצויים במיקום הרצוי.

חוצץ המסך המוקדש למוד טקסט, מתחיל בכתובת b8000H וגודלו 4000 בתים. מבנה החוצץ זהה למבנה החוצץ שהגדרנו בסעיף 7.4. לכל משבצת תו במסך מוקצים שני בתים: אחד מכיל את ערך התו, והשני את מאפייני הצבע. בתחילת הפרק הזכרנו שבמוד טקסט המסך נמצא ברזולוציה של 25x80, כלומר הוא מייצג 2000 **משבצות**, או תאים, שעבור כל אחת מהן מוגדרים תו ומאפייני צבע. כל משבצת כזו מייצגת מטריצה של נקודות מסך (pixels).

מכיון שאנו עוסקים בזיכרון, כדאי שנלמד כיצד מורכב הבית בו שמורים מאפייני הצבע. הבית מורכב משמונה סיביות, המחולקות לקבוצות שלכל אחת מהן יש תפקיד (שימו לב ש-b0 הינה הסיבית הימנית ביותר ו-b7 היא השמאלית ביותר):

- b0-b3 מכילות את צבע הטקסט (0-15).
- b4-b6 מכילות את צבע הרקע (0-7).
- ו-b7 היא סיבית ההבהוב (סיבית מורמת מסמלת הבהוב).

מתוך כך, ניתן להבין מדוע כשרצינו להוסיף הבהוב, חיברנו לצבע המבוקש 128 (10000000 בינארי).

עכשיו, כשאנו יודעים כיצד בנוי חוצץ מסך הטקסט, אפשר להתחיל לשתול בו ערכים, או לקרוא ממנו. תחילה נגדיר מצביע אל תחילת החוצץ. כיון שכתובת תחילת החוצץ אינה נמצאת באזור הזיכרון של התוכנית, עלינו להגדיר את המצביע כמצביע רחוק (**far pointer**) לסגמנט b800H עם היסט 0 (כלומר כתובת b8000H). נעשה זאת כך:

```
char far *TextScreen = MK_FP (0xb800,0);
```

חוצץ המסך הוא ליניארי ולכן, כדי להגיע לנקודה (x,y) עלינו לגשת להשתמש בהיסט:

```
( (y-1)*WIDTH+(x-1) )*2
```

הורדתי 1 מערכי ה-x וה-y, כי המשבצת הראשונה נמצאת בהיסט 0 (אפס), כפלתי הכל ב-2, כי אמרנו שעבור כל משבצת במסך שמורים שני בתים.

אם כך, פונקציה לשתילת תו במיקום x,y על פי מאפייני צבע עשויה להיראות כך:

```
void CPutChar (int x, int y, char ch, char attrib)
{
 int Pos = ((y-1)*WIDTH+(x-1) )*2;
 TextScreen[Pos] = ch;
 TextScreen[Pos+1] = attrib;
}
```

אם אתם תוהים מדוע התייחסתי לחוצץ זיכרון המסך כמערך וכתבתי את ההיסט בסוגריים מרובעים (במקום להזיז את המצביע), הרי התשובה: עשיתי זאת בדיוק מאותה סיבה שבגללה הגדרתי בסעיף 7.4 את חוצץ הזיכרון כמחרוזת בגודל 4000 (במקום להגדיר מצביע ולהקצות לו 4000 תאים בעזרת malloc). עשיתי זאת בהסתמך על הגדרת המערך: "מערך הוא אוסף רציף של נתונים מסוג זהה. שם המערך הוא מצביע לאיבר הראשון בו", כפי שהוסבר בפרק הראשון. תכונה זו של מערך אינה ידועה לרבים מהמתכנתים (בעיקר לאלה שלמדו פסקל לפני לימוד C). בדרך כלל נוהגים להזיז את המצביע על פני החוצץ, במקום להתייחס לחוצץ כמערך שבו תזוזת המצביע הינה טבעית. אנא קראו את הפרק הראשון, אם הסבר זה אינו מספיק.

עכשיו, כשאנו יודעים כיצד להגיע לכל משבצת במסך, נוכל לבצע את הפעולות הבאות:

- שתילת תו במקום מסוים בצבע מסוים.
- קריאת התו הנמצא במיקום מסוים.
- קריאת מאפייני הצבע של תו הנמצא במיקום מסוים.
- שינוי מאפייני הצבע של תו הנמצא במיקום מסוים.
- העתקת אוסף תווים שאינו מלבני (לדוגמה, מתחילת השורה הראשונה עד אמצע שורה שלישית). מכיוון שהחוצץ הוא ליניארי, נעתיק את כל מה שנמצא בין שתי נקודות רצויות.

התוכנית הבאה מדגימה מספר פעולות של גישה ישירה לזיכרון:

```
/*** TextScr.C ***/

#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define WIDTH  80
#define LENGTH 25
char far *TextScreen = MK_FP (0xb800,0);
```

```
void PutChar (int x, int y, char ch)
///// פונקציה לשתילת תו במיקום (x,y)
{
   TextScreen[  ( (y-1)*WIDTH+(x-1) )*2  ] = ch;
}

void CPutChar (int x, int y, char ch, char attrib)
///// פונקציה לשתילת תו במיקום (x,y) בצבע מבוקש
{
   int Pos= ((y-1)*WIDTH+(x-1) )*2;
   TextScreen[Pos] = ch;
   TextScreen[Pos+1] = attrib;
}
char GetChar (int x, int y)
///// פונקציה לקריאת תו ממיקום (x,y)
{
   return (TextScreen[  ( (y-1)*WIDTH+(x-1) )*2  ]);
}
char GetAttrib (int x, int y)
///// פונקציה לקריאת מאפייני הצבע ממיקום (x,y)
{
   return (TextScreen[  ( (y-1)*WIDTH+(x-1) )*2 +1 ]);
}

void ChangeAttrib (int x, int y, int Attrib)
///// פונקציה לשינוי מאפייני הצבע במיקום (x,y)
{
   TextScreen[  ( (y-1)*WIDTH+(x-1) )*2 +1 ] = Attrib;
}

void main (void)
{
   int i;
   clrscr();                          // שתילת תו במיקום (1,1)
   PutChar (1,1,1);
   getch();
   CPutChar(2,1,GetChar(1,1),128+5); // העתקת התו שב(1,1) ל(2,1) בסגול מהבהב
   getch();
   ChangeAttrib(1,1,GetAttrib(2,1)); // שינוי הצבע של (1,1) על פי הצבע של (2,1)
   getch();
   for (i=1;i<=80;i++)               // כתיבת שורה של תווים בשורה 10
      PutChar(i,10,i);
   getch();
```

```
   for ( i=((10-1)*WIDTH+(10-1))*2+1; i<=((10-1)*WIDTH+(70-1))*2+1; i+=2 )
   {
      // שינוי מאפייני הצבע של חלק מהשורה לירוק מהבהב
      // ממיקום (10,10) עד מיקום (70,10)
      TextScreen[i]=128+2;
      delay(100);
   }
}
```

בפרק הבא, נלמד כיצד לשלב בתוכנית עכבר שיתמוך במוד טקסט.

פרק 8

עכבר למסך טקסט

בפרק זה נלמד להפעיל את העכבר ולהשתמש בו.

- ✓ **נלמד להשתמש בפונקציות שונות של פסיקת השירות 33H – פסיקת העכבר.**
- ✓ **נלמד לעקוב אחר תנועת העכבר.**
- ✓ **נלמד לעקוב אחר לחיצות בלחצני העכבר.**
- ✓ **נלמד לתחום את שטח תנועת העכבר.**

השימוש בעכבר, המוסבר בפרק זה מיועד לעבודה במוד טקסט. בפרקים הבאים נכיר גם מודים אחרים וניצור להם יחידה נפרדת.

השימוש בעכבר מתבצע בעזרת פסיקת העכבר 33H, המכילה שירותים שונים לתפעול העכבר. כדי להפעיל שירות מסוים, נכניס לפני הפעלת הפסיקה אל האוגר ax את קוד השירות שישמש כפרמטר. את הפסיקה נפעיל בעזרת הפונקציה int86, כפי שלמדנו בפרק 3.

8.1 זיהוי העכבר

כל תוכנית המשתמשת בעכבר, חייבת לבדוק תחילה אם העכבר מותקן במחשב. כדי לעשות זאת, נשתמש בפסיקת העכבר (33H) עם פרמטר 0.

לאחר הפעלת הפסיקה, ערכי האוגרים ישתנו כך:

- אוגר ax יכיל "אמת", אם נמצא עכבר מותקן ומחובר במחשב.
- אוגר bx יכיל את מספר הלחצנים שבעכבר.

אם כך, ניתן בקלות לכתוב פונקציה לזיהוי העכבר, אשר תנצל את הפונקציה int86, כפי שלמדנו בפרק 3.

הפונקציה תיראה כך:

```
int CheckMouse ( void )
{
  union REGS ireg ;

  ireg.x.ax =0x00 ;
  int86 ( 0x33, &ireg , &ireg ) ;
  if (ireg.x.ax)          // אם נמצא עכבר, החזר את מספר הלחצנים שלו
    return (ireg.x.bx);
  else                          // אחרת החזר אפס
    return (0);
}
```

8.2 הצגה והסתרה של המצביע

פסיקת העכבר עם פרמטרים 1 ו-2 משמשת להצגה ולהסתרה של מצביע העכבר. כאשר המצביע במצב "מוסתר" – איננו רואים אותו, אך פעולת העכבר ממשיכה כסדרה. כלומר, אם נזיז את העכבר או נלחץ על לחצניו, התוכנית תגיב לפעולות אלו כרגיל, אך לא נראה דבר על המסך.

מתי נרצה להסתיר את מצביע העכבר?

בכל פעם שעורכים שינויים בתוכן המסך, אנו עלולים לגרום לעיוותים שונים. לדוגמה, אם נכתוב בשורה האחרונה של המסך הודעה שבסופה תווי הבקרה "n\", המחשב יגלגל את תוכן המסך כלפי מעלה והמצביע יעלה שורה אחת, למרות שזה אינו מיקומו האמיתי. (המצביע צריך להישאר בשורת המסך האחרונה).

תופעות אלו נגרמות, מכיון שהמצביע מצויר על המסך מחדש בכל פעם שיש שינוי בקואורדינטות שלו. למרות שהעכבר לא זז, המצביע שלו הוזז כלפי מעלה יחד עם כל תוכן המסך.

אם כך, לפני שמבצעים שינויים בתוכן המסך עלינו להסתיר את המצביע, ומייד בסוף הפעולה עלינו להציג אותו מחדש.

פונקציה להצגת המצביע, תיראה כך:

```
void ShowMouse ( void )
{
  union REGS ireg ;

  ireg.x.ax = 0x01 ;
  int86 ( 0x33 , &ireg , &ireg ) ; //  קריאה לפסיקה 33 עם הפרמטר 1
}
```

בפונקציה להסתרת המצביע נעשה פעולה זהה עם הפרמטר 2.

8.3 בדיקת סטטוס העכבר

לאחר שלמדנו כיצד לזהות את העכבר ולהציג את המצביע שלו על המסך, עלינו לבדוק מהו מיקום העכבר ומהו מצב הלחצנים שלו.

כדי לעשות זאת, נשתמש בפסיקת העכבר (33H) עם פרמטר 3.

בעקבות הפעלת הפסיקה, ערכי האוגרים ישתנו כך:

- אוגר bx יכיל את מצב הלחצנים.
- אוגר cx יכיל את המיקום האופקי של העכבר.
- אוגר dx יכיל את המיקום האנכי של העכבר.

בדיקת מצב לחצני העכבר

לאחר הפעלת הפסיקה יתקבל באוגר bx מצב הלחצנים כך:

- ערך סיבית 0 יהיה **אמת**, אם הלחצן השמאלי לחוץ.
- ערך סיבית 1 יהיה **אמת**, אם הלחצן הימני לחוץ.
- ערך סיבית 2 יהיה **אמת**, אם הלחצן האמצעי לחוץ.

אם כך, נוכל לחלק את מצב הלחצנים לשמונה מקרים שונים (ראה תרשים 8.1).

b3 b2	b1	b0	bx
א	י	ש	
			0
		+	1
	+		2
	+	+	3
+			4
+		+	5
+	+		6
+	+	+	7

ש - שמאלי
י - ימני
א - אמצעי
+ - לחוץ

תרשים 8.1

המרת קואורדינטות העכבר למסך הממשי

המסך מיוצג כמסך וירטואלי ברזולוציה של 640x200, כך שמיקום הנקודה השמאלית-העליונה הוא (0,0), ומיקום הנקודה הימנית-התחתונה הוא (639,199).

המסך הממשי במצב טקסט הוא ברזולוציה 80 x25 וכך, מיקום הנקודה השמאלית-העליונה הוא (0,0), ומיקום הנקודה הימנית-התחתונה הוא (79,24).

לכן, עלינו להמיר את הערכים המתקבלים, כתוצאה מהפעלת הפסיקה באוגרים cx ו-dx לערכים שיתאימו למסך הממשי שלנו. נעשה זאת על ידי חלוקה בשמונה של המיקום האופקי ושל המיקום האנכי.

למה דווקא שמונה (8)? נחשב זאת: 640/8 שווה 80 ו- 200/8 שווה 25, וכך מומרים ערכי הפיקסלים לתאי התווים שבמסך הממשי. לדוגמה, כל הערכים בתחום ריבוע הפיקסלים (0,0) עד (7,7) ייצגו את המשבצת (0,0) במסך הממשי.

על כן, הפונקציה לבדיקת סטטוס העכבר תיראה כך:

```
int MouseStatus(int *X,int *Y)
{
  union REGS ireg ;

  ireg.x.ax =0x03 ;
  int86 ( 0x33, &ireg , &ireg ) ; //  קריאה לפסיקה 33 עם הפרמטר 3

  *X = ireg.x.cx/8 ;
  *Y = ireg.x.dx/8 ;

  return(ireg.x.bx) ;          //  החזר את מצב הלחצנים
}
```

8.4 מציאת מיקום לחיצה/שחרור אחרון

לעיתים נרצה לבצע פעולות גרירה שונות. למטרות אלו נוכל להיעזר בפונקציה הבודקת מה היה המיקום האחרון בו לחצן מסוים נלחץ או שוחרר. לשם כך נשתמש בפסיקת העכבר עם הפרמטרים 5 ו-6 (5 - לחיצה, 6 - שחרור).

לפני הפעלת הפסיקה:

- אוגר bx צריך להכיל את מספר הלחצן שאותו רוצים לבדוק.

לאחר הפעלת הפסיקה:

- אוגר cx יכיל את המיקום האופקי של הלחיצה/שחרור אחרון של הלחצן המבוקש.
- אוגר dx יכיל את המיקום האופקי של הלחיצה/שחרור אחרון של הלחצן המבוקש.

שימו לב! הפעם, לחצני העכבר מיוצגים בדרך שונה: 0 - שמאלי, 1 - ימני, 2 - אמצעי (בניגוד לסימולים 1, 2, 4 על פי תרשים 8.1).

הפונקציה לבדיקת מיקום הלחיצה האחרונה של לחצן תיראה כך:

```
void MousePressPos ( int Button , int *X , int *Y )
{
  union REGS ireg ;

  ireg.x.ax =0x05 ;
  ireg.x.bx = Button ;
  int86 ( 0x33, &ireg , &ireg ) ; // קריאה לפסיקה 33 עם הפרמטר 5

  ///// עדכון הקואורדינטות
  *X = ireg.x.cx/8 ;
  *Y = ireg.x.dx/8 ;
}
```

בפונקציה למציאת מיקום השחרור, נעשה פעולה דומה עבור הפרמטר 6 במקום פרמטר 5.

8.5 תיחום מרחב תנועת העכבר

לעיתים נרצה להגביל את תנועת העכבר רק לחלק מהמסך, כדי למנוע גישה של המשתמש אל אזורים מסוימים. כדי לתחום את מרחב תנועת העכבר נשתמש בפסיקת העכבר עם הפרמטרים 7 ו-8 (7 - תיחום אופקי, 8 - תיחום אנכי).

לפני הפעלת הפסיקה:

- אוגר cx צריך להכיל את הערך המינימלי של התחום (אופקי או אנכי) על המסך.
- אוגר dx צריך להכיל את הערך המקסימלי של התחום (אופקי או אנכי) על המסך.

בתיחום אופקי, הערך המינימלי של התחום הוא מספר הטור שיציין את הגבול השמאלי על המסך, מיקום הערך המקסימלי הוא מספר הטור שיציין את הגבול הימני על המסך.

בתיחום אנכי, הערך המינימלי של התחום הוא מספר הטור שיציין את הגבול העליון במסך, מיקום הערך המקסימלי הוא מספר הטור שיציין את הגבול התחתון במסך.

שימו לב! יש להמיר את ערכי המיקום מערכים ממשיים לערכים וירטואליים, פירוש הדבר - לכפול ערכים האלה ב-8.

לאחר תיחום מרחב תנועת העכבר, עלינו לדאוג שמיקומו הנוכחי יהיה בתוך התחום המוגדר בלבד. לשם כך נזיז אותו אל הפינה השמאלית-העליונה של התחום, בעזרת פסיקת העכבר עם הפרמטר 4.

לפני הפעלת הפסיקה:

- אוגר cx צריך להכיל את המיקום האופקי שאליו אנו רוצים להזיז את העכבר.
- אוגר dx צריך להכיל את המיקום האנכי שאליו אנו רוצים להזיז את העכבר.

פונקציית התיחום תיראה כך:

```
int MouseRange ( int MinX, int MinY, int MaxX, int MaxY )
{
  union REGS ireg ;
  int Err = 0 ;                 // האם התגלתה תקלה

  ///// בדיקת חריגות בערכים , והמרה לערכי המסך הווירטואלי /////
  if ( MinX >=0 && MinX < 80 )
    MinX *= 8 ;
  else Err = 1 ;

  if ( !Err &&  MaxX >=0 && MaxX < 80 )
    MaxX *= 8 ;
  else Err = 1 ;

  if ( !Err &&  MinY >=0 && MinY < 25 )
    MinY *= 8 ;
  else Err = 1 ;

  if ( !Err &&  MaxY >=0 && MaxY < 25 )
    MaxY *= 8 ;
  else Err = 1 ;

  if (Err) return (0) ;               // יציאה אם יש תקלה

  ///// הגדרת התחום האופקי /////
  ireg.x.ax=0x7 ;
  ireg.x.cx = MinX ;
  ireg.x.dx = MaxX ;
  int86 ( 0x33 , &ireg , &ireg ) ;  //  קריאה לפסיקה 33 עם הפרמטר 7

  ///// הגדרת התחום האנכי /////
  ireg.x.ax=0x8  ;
  ireg.x.cx = MinY ;
  ireg.x.dx = MaxY ;
  int86 ( 0x33 , &ireg , &ireg ) ; // קריאה לפסיקה 33 עם הפרמטר 8
```

```
  ///// הזזת הסמן לפינה השמאלית עליונה של התחום
  ireg.x.ax = 0x04 ;
  ireg.x.cx = MinX ;
  ireg.x.dx = MinY ;
  int86( 0x33 , &ireg , &ireg ) ; // קריאה לפסיקה 33 עם הפרמטר 4

  return (1) ;
}
```

כדאי שנכין גם פונקציה לביטול התיחום הנוכחי. נעשה זאת על ידי הפעלת פונקציית התיחום על קואורדינטות המסך הממשי.

```
MouseRange(0,0,79,24) ;
```

8.6 יחידה לשימוש בעכבר

ממשק היחידה

```
/*************************************************************************
 *                          M o u s e T . H                              *
 *                         ~~~~~~~~~~~~~~~~~~                            *
 *             ממשק ליחידה לטיפול ושימוש בעכבר במסך טקסט                 *
 *************************************************************************/

//////////////////////////////////////////////////////////////////////////
//                        ה צ ה ר ו ת                                   //
//////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <conio.h>
#include <dos.h>

//////////////////////////////////////////////////////////////////////////
//                      פ ו נ ק צ י ו ת                                 //
//////////////////////////////////////////////////////////////////////////

int CheckMouse ( void );
/*------------------------------------------------------------------------
    פונקציה הבודקת האם מותקן עכבר במחשב, האם הוא מחובר וכמה לחצנים יש לו.
    פרמטרים מועברים - אין.
    ערך מוחזר - אפס אם לא נמצא עכבר מחובר ומותקן,
    אחרת יוחזר מספר הלחצנים בעכבר.
 ------------------------------------------------------------------------*/
```

```
void ShowMouse ( void );
/*---------------------------------------------------------------------------
   פונקציה ההופכת את מצביע העכבר לנראה (Visible).
   פרמטרים מועברים - אין.
   ערך מוחזר - אין.
 ---------------------------------------------------------------------------*/

void HideMouse ( void );
/*---------------------------------------------------------------------------
   פונקציה המחביאה את מצביע העכבר.
   פרמטרים מועברים - אין.
   ערך מוחזר - אין.
 ---------------------------------------------------------------------------*/

int MouseRange ( int MinX, int MinY, int MaxX, int MaxY );
/*---------------------------------------------------------------------------
   פונקציה להגדרת מרחב פעולת העכבר.
   פרמטרים מועברים:
   MinX , MinY - קואורדינטות נקודת הגבול המינימלי
   MaxX , MaxY - קואורדינטות נקודת הגבול המקסימלי
   ערך מוחזר: "אמת" אם התיחום התבצע בהצלחה.
 ---------------------------------------------------------------------------*/

void CancelRange ( void );
/*---------------------------------------------------------------------------
   פונקציה לביטול התחום הנוכחי של העכבר.
   פרמטרים מועברים - אין.
   ערך מוחזר - אין.
 ---------------------------------------------------------------------------*/

int MouseStatus(int *X,int *Y);
/*---------------------------------------------------------------------------
   פונקציה לבדיקת סטטוס העכבר ומיקומו.
   פרמטרים מועברים :
   X (By Address) - יקבל את מספר הטור עליו עומד המצביע.
   Y (By Address) - יקבל את מספר השורה עליו עומד המצביע.
   הפונקציה תחזיר את אחד מהערכים הבאים:
   אף לחצן לא לחוץ - 0
   לחצן שמאלי - 1
   לחצן ימני - 2
   לחצן שמאלי + ימני - 3
   לחצן אמצעי - 4
   לחצן שמאלי ואמצעי - 5
```

```
                                         לחצן ימני ואמצעי - 6
                                     שלושת הלחצנים לחוצים - 7
 ---------------------------------------------------------------------------*/

void MousePressPos ( int Button , int *X , int *Y );
/*--------------------------------------------------------------------------
                               פונקציה למציאת מיקום הלחיצה האחרונה.
                                                   פרמטרים מועברים:
             Button - איזה לחצן לבדוק (שמאלי-0, ימני-1, אמצעי-2)
                               X (By Address) - יקבל את מספר הטור
                             Y (By Address) - יקבל את מספר השורה.
                                                    ערך מוחזר - אין.
 ---------------------------------------------------------------------------*/

void MouseReleasePos ( int Button , int *X , int *Y );
/*--------------------------------------------------------------------------
                               פונקציה למציאת מיקום השחרור האחרון.
                                                   פרמטרים מועברים:
             Button - איזה לחצן לבדוק (שמאלי-0, ימני-1, אמצעי-2)
                               X (By Address) - יקבל את מספר הטור
                             Y (By Address) - יקבל את מספר השורה.
                                                    ערך מוחזר - אין.
 ---------------------------------------------------------------------------*/
```

מימוש היחידה

```
/***************************************************************************
 *                              M o u s e T . C                            *
 *                             ~~~~~~~~~~~~~~~~~~                          *
 *                 יחידה לטיפול ושימוש בעכבר במסך טקסט                     *
 ***************************************************************************/
////////////////////////////////////////////////////////////////////////////
//                          ה צ ה ר ו ת                                   //
////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <dos.h>

int NumOfButtons   ; // מספר הלחצנים בעכבר
```

```
////////////////////////////////////////////////////////////////////////////
//                        פ ו נ ק צ י ו ת                                 //
////////////////////////////////////////////////////////////////////////////

int CheckMouse ( void )
/*--------------------------------------------------------------------------
   פונקציה הבודקת האם מותקן עכבר במחשב, האם הוא מחובר, וכמה לחצנים יש לו.
   פרמטרים מועברים - אין.
   ערך מוחזר - אפס אם לא נמצא עכבר מחובר ומותקן,
               אחרת יוחזר מספר הלחצנים בעכבר.
 --------------------------------------------------------------------------*/
{
   union REGS ireg ;
   ireg.x.ax =0x00 ;
   int86 ( 0x33, &ireg , &ireg ) ;  // קריאה לפסיקה עם פרמטר 0
      // כתוצאה מהפסיקה, אוגר AX יכיל "אמת" אם יש עכבר מחובר ומותקן במחשב,
      // ואוגר BX יכיל את מספר הלחצנים של העכבר המחובר.

   if (ireg.x.ax)                    // אם נמצא עכבר החזר את מספר הלחצנים
      return (ireg.x.bx);
   else                              // אחרת החזר אפס
      return (0);
}

void ShowMouse ( void )
/*--------------------------------------------------------------------------
   פונקציה ההופכת את מצביע העכבר לנראה (Visible).
   פרמטרים מועברים - אין.
   ערך מוחזר - אין.
 --------------------------------------------------------------------------*/
{
   union REGS ireg ;
   ireg.x.ax = 0x01 ;
   int86 ( 0x33 , &ireg , &ireg ) ; // קריאה לפסיקה 33 עם פרמטר 1
}
```

```
void HideMouse ( void )
/*--------------------------------------------------------------------------
                                        פונקציה המחביאה את מצביע העכבר.
                                                    פרמטרים מועברים - אין.
                                                          מוחזר - אין ערך.
 -------------------------------------------------------------------------*/
{
   union REGS ireg ;
   ireg.x.ax = 0x02 ;
   int86 ( 0x33 , &ireg , &ireg ) ; // קריאה לפסיקה 33 עם פרמטר 2
}

int MouseRange ( int MinX, int MinY, int MaxX, int MaxY )
/*--------------------------------------------------------------------------
                                        פונקציה להגדרת מרחב פעולת העכבר.
                                                        פרמטרים מועברים :
                       קואורדינטות נקודת הגבול המינימלי - MinX , MinY
                       קואורדינטות נקודת הגבול המקסימלי - MaxX , MaxY
                                ערך מוחזר : "אמת" אם התיחום התבצע בהצלחה.
 -------------------------------------------------------------------------*/
{
   union REGS ireg ;
   int Err = 0 ; // האם התגלתה תקלה

   ///// בדיקת חריגות בערכים, והמרה לערכי המסך הווירטואלי
   if ( MinX >=0 && MinX < 80 )
      MinX *= 8 ;
   else Err = 1 ;

   if ( !Err &&  MaxX >=0 && MaxX < 80 )
      MaxX *= 8 ;
   else Err = 1 ;

   if ( !Err &&  MinY >=0 && MinY < 25 )
      MinY *= 8 ;
   else Err = 1 ;

   if ( !Err &&  MaxY >=0 && MaxY < 25 )
      MaxY *= 8 ;
   else Err = 1 ;
```

```
   if (Err) return (0) ; // יציאה אם יש תקלה

   ///// הגדרת התחום האופקי
   ireg.x.ax=0x7 ;
   ireg.x.cx = MinX ;
   ireg.x.dx = MaxX ;
   int86 ( 0x33 , &ireg , &ireg ) ;  // קריאה לפסיקה 33 עם פרמטר 7

   ///// הגדרת התחום האנכי
   ireg.x.ax=0x8  ;
   ireg.x.cx = MinY ;
   ireg.x.dx = MaxY ;
   int86 ( 0x33 , &ireg , &ireg ) ;  // קריאה לפסיקה 33 עם פרמטר 8

   ///// הזזת הסמן לפינה השמאלית עליונה של התחום
   ireg.x.ax = 0x04 ;
   ireg.x.cx = MinX ;
   ireg.x.dx = MinY ;
   int86( 0x33 , &ireg , &ireg ) ;   // קריאה לפסיקה 33 עם פרמטר 4

   return (1) ;    // יציאה מהפונקציה
}

void CancelRange ( void )
/*-------------------------------------------------------------------------
        פונקציה לביטול התחום הנוכחי של העכבר.
        פרמטרים מועברים - אין.
        ערך מוחזר - אין.
 -------------------------------------------------------------------------*/
{
   MouseRange(0,0,79,24) ; // הפעלת פונקציית ההגבלה עם קואורדינטות גבולות המסך
}

int MouseStatus(int *X,int *Y)
/*-------------------------------------------------------------------------
        פונקציה לבדיקת סטטוס העכבר ומיקומו.
        פרמטרים מועברים :
          X (By Address) - הטור עליו עומד המצביע יקבל מספר.
          Y (By Address) - יקבל את מספר השורה עליו עומד המצביע.
        הפונקציה תחזיר את אחד מהערכים הבאים :
          אף לחצן לא לחוץ - 0
          לחצן שמאלי - 1
          לחצן ימני - 2
```

```
                                   לחצן שמאלי וימני - 3
                                          לחצן אמצעי - 4
                                   לחצן שמאלי ואמצעי - 5
                                    לחצן ימני ואמצעי - 6
                              שלושת הלחצנים לחוצים - 7
 ---------------------------------------------------------------------------*/
{
   union REGS ireg ;
   ireg.x.ax =0x03 ;
   int86 ( 0x33, &ireg , &ireg ) ; // קריאה לפסיקה 33 עם פרמטר 3
   *X = ireg.x.cx ;
   *Y = ireg.x.dx ;

   ///// המרה ממסך וירטואלי לממשי
   *X /= 8 ;
   *Y /= 8 ;
   return(ireg.x.bx) ;        // החזר סטטוס עכבר
}

void MousePressPos ( int Button , int *X , int *Y )
/*---------------------------------------------------------------------------
                           פונקציה למציאת מיקום הלחיצה האחרונה.
                                                   פרמטרים מועברים :
           Button - (אמצעי-2,ימני-1,שמאלי-0) איזה לחצן לבדוק.
                         X (By Address) - יקבל את מספר הטור
                        Y (By Address) - יקבל את מספר השורה.
                                                      ערך מוחזר - אין.
 ---------------------------------------------------------------------------*/
{
   union REGS ireg ;
   ireg.x.ax =0x05 ;
   ireg.x.bx = Button ;
   int86 ( 0x33, &ireg , &ireg ) ; // קריאה לפסיקה 33 עם פרמטר 5

   ///// עדכון הקואורדינטות
   *X = ireg.x.cx ;
   *Y = ireg.x.dx ;

   ///// המרה ממסך וירטואלי לממשי
   *X /= 8 ;
   *Y /= 8 ;
}
```

```
void MouseReleasePos ( int Button , int *X , int *Y )
/*------------------------------------------------------------------------
                    פונקציה למציאת מיקום השחרור האחרון.
                    פרמטרים מועברים :
          Button - (אמצעי-2,ימני-1,שמאלי-0) איזה לחצן לבדוק.
          X (By Address) - יקבל את מספר הטור
          Y (By Address) - יקבל את מספר השורה.
                    ערך מוחזר - אין.
 ------------------------------------------------------------------------*/
{
   union REGS ireg ;
   ireg.x.ax =0x06 ;
   ireg.x.bx = Button ;
   int86 ( 0x33, &ireg , &ireg ) ; // 6 קריאה לפסיקה 33 עם פרמטר

   ///// עדכון הקואורדינטות
   *X = ireg.x.cx ;
   *Y = ireg.x.dx ;

   ///// המרה ממסך וירטואלי לממשי
   *X /= 8 ;
   *Y /= 8 ;
}
```

תוכנית לדוגמה

לאחר כל לחיצה על אחד מלחצני העכבר, התוכנית מציגה על המסך את סימול הלחצן שנלחץ ואת מיקום הלחיצה.

התוכנית תסתיים בהקשה על מקש Esc.

על פי תוכנית זו ניתן לדעת כיצד ליצור תוכניות שפועלות עם עכבר ומקלדת במקביל, וכיצד לזהות את מיקום הלחיצות. לשם כך צריך להחליף את הדפסת ההודעה בביטוי case הבודק את מיקום הלחיצה, ומבצע את הפעולות הרצויות על פי מיקום הלחיצה.

הקבצים שיש לצרף לפרויקט כדי להריץ את תוכנית הדוגמה הם : Mouset.c ו-Tmousex.c.
למידע נוסף קראו את נספח ו'.

```
/*** TMousex.C ***/

#include <stdio.h>
#include <conio.h>
#include "MouseT.h"
```

```
void main (void)
{
   int Quit=0,x,y,s=0;
   char c;

   if (!CheckMouse())                //בדיקת העכבר
   {
      printf("Mouse Not Found!");
      exit (1);
   }
   ShowMouse();                      // הצג את מצביע העכבר
   while  (!Quit)                    // כל עוד לא נלחץ Esc
   {
      while ( !kbhit() && !s )       // כל עוד לא נלחץ עכבר או מקש
         s=MouseStatus(&x,&y);
      while ( !kbhit() && MouseStatus(&x,&y) );  //כל עוד הלחצן לחוץ
      if   (s)
      {
         HideMouse();
         printf ("הלחצן שסימונו %d נלחץ במיקום (%d,%d)\n",x,y,s);
         ShowMouse();
         s=0;
      }
      else
      {
         c = getch();
         Quit = (c == 27);  // אמת אם הלחצן שנלחץ הוא Esc
      }
   }
}
```

שימו לב! לפני ואחרי הצגת ההודעה על המסך, מצביע העכבר מוסתר ומוצג מחדש.

גם אם לוחצים על יותר מלחצן אחד, ההודעה תציג סימול לחצן, המראה שרק אחד מהלחצנים נלחץ. הסיבה לכך היא, שגם כשנדמה שלוחצים על שני לחצנים יחד, יש לחצן שנלחץ ראשון והוא זה שסימונו יוצג במסך. גם אם סימון הלחצנים שיוצגו על המסך יהיה הסטטוס האחרון של העכבר לפני השחרור, הבעיה לא תיפתר כי תמיד לחצן אחד משוחרר לפני הלחצן האחר. אם רוצים לבצע פעולות במקרה של לחיצה על יותר מלחצן אחד, צריך לשלוף את סטטוס העכבר באמצע הלחיצה, ואז הסימון יהיה נכון. אבל כיצד ניתן לעשות זאת?

כיון שאיננו יכולים לנחש מהו זמן אמצע הלחיצה, נצטרך לעשות זאת בשיטה אחרת. נוכל למשל, להכניס ללולאה השנייה (שמחכה לשחרור), פקודות לבדיקה חוזרת של מצב שלוש הסיביות הראשונות של ערך סימון הלחצנים, שהתקבל מהפעלת פונקציית סטטוס העכבר, ולהשוות אותם לשלוש הסיביות הראשונות במשתנה אחר. אם ערך הסיבית בסטטוס הנוכחי הוא אמת בעוד שערך הסיבית המתאימה לה במשתנה שלנו הוא שקר, נהפוך את הערך למצב אמת. באופן זה, המשתנה יכיל בסופו של דבר את הסימון של מירב הלחצנים שנלחצו.



נסו לכתוב תוכנית שעושה זאת.
תוכלו להיעזר בפרק 2, "עבודה בסיביות".

העתקה והדבקה

כיצד נוכל לבצע פעולות העתקה והדבקה?

כדי לבצע העתקה, עלינו לסמן תחילה את התחום שברצוננו להעתיק. לשם כך נבדוק מה היה המיקום בתחילת הלחיצה ובשחרור הלחיצה. כדי להעתיק את קטע הטקסט, עלינו לפתוח חוצץ בזיכרון ולהעתיק לתוכו את הקטע המתאים מתוך חוצץ מסך הטקסט (ראו פרק קודם). כדי להדביק, פשוט נעתיק את תוכן החוצץ שיצרנו אל המקום המבוקש בחוצץ מסך הטקסט.

שימו לב! הסימון לא צריך להיות של מלבן, אלא של התווים המבוקשים בלבד (לדוגמה, סימון יכול להתחיל מאמצע שורה 0 עד אמצע שורה 3). מכיון שחוצץ מסך הטקסט הוא ליניארי (כמו שלמדנו בפרק הקודם), בעיה זו נפתרת מעצמה.



נסו לכתוב מעבד תמלילים בסיסי המשלב פעולות של העתקה והדבקה.

פרק 9

גרפיקה ב- 256 צבעים (Mode 13H)

בפרק זה נכיר את מוד 13H של VGA וניצור יחידת ספריה לפעולות גרפיקה שונות ב-256 צבעים.

- ✔ **נלמד להחליף בין המודים השונים של המסך.**
- ✔ **נלמד לשתול פיקסלים על המסך.**
- ✔ **נלמד לצייר צורות בסיסיות שונות על המסך כגון קווים, מלבנים ועיגולים.**
- ✔ **נלמד להשתמש בלוח הצבעים וניצור אפקטים שונים בעזרתו.**
- ✔ **נלמד ליצור דפים וירטואליים שישפרו את מהירות ואיכות התוכנית.**
- ✔ **לסיום, נלמד ליצור גופנים כדי לשלב טקסט במסך גרפי.**

9.1 מה זה בכלל VGA Mode 13H?

VGA הם ראשי התיבות של Video Graphics Array – מתאם מסך המתקשר בין המחשב לבין המסך המחובר אליו. כיום, כמעט כל כרטיס מסך תומך ב-VGA. שיטת VGA מאפשרת עבודה במודים (אופנים) שונים, שכל אחד מהם פועל במספר צבעים שונה וברזולוציה שונה.

יחידת הספריה הגרפית של שפת Graphics.h) C) אינה בנויה לפעולה במצב של 256 צבעים, לכן עלינו ליצור את יחידת הספריה בעצמנו. מוד 13H מאפשר הצגת 256 צבעים שונים בו-זמנית ברזולוציה של 320X200 פיקסלים. קואורדינטות הפינה השמאלית העליונה הם (0,0), וקואורדינטות הפינה הימנית התחתונה הם (199,131).

מוד זה פועל עם חוצץ (buffer) ליניארי בגודל 64KB, כך שכל פיקסל על המסך מיוצג על ידי בית אחד (64000=200*320). הגודל הקטן של החוצץ מקל על התכנות ומתאים לשימוש בשפות מחשב של 16 סיביות, כמו שפת C.

אם פיסקה זו נשמעת לכם כמו סינית, אל דאגה, הדברים יתבהרו במהלך הלימוד.

9.2 החלפה בין מודים

כדי לקבוע את המוד הרצוי יש להשתמש בפסיקת המסך 10H. אוגר ah צריך להכיל את הערך אפס ואוגר al צריך להכיל את מספר המוד הרצוי.

אם כן, ניתן בקלות לקבוע את מוד 13H על ידי שימוש בפונקציה int86, כפי שלמדנו בפרק 3.

```
void Set256Mode (void)
{
  union REGS regs;
  regs.h.ah = 0 ;
  regs.h.al = 0x13 ;
  int86 ( 0x10 , &regs , &regs ) ;
}
```

שימו לב! בסוף התוכנית חייבים לחזור למוד טקסט (3H), אחרת עלול להיגרם נזק למסך. לשם כך, נכתוב פרוצדורה שתחזיר אותנו למוד טקסט:

```
void SetTextMode (void)
{
  union REGS regs ;

  regs.h.ah = 0 ;
  regs.h.al = 0x3 ;
  int86 ( 0x10 , &regs , &regs ) ;
}
```

9.3 שתילת פיקסל על המסך

הדרך הפשוטה ביותר לשתילת פיקסל על המסך היא על ידי הפונקציה cH של פסיקת המסך 10H. לשם כך, אוגר ah צריך להכיל את מספר הפונקציה (cH), אוגר al יכיל את הצבע המבוקש, אוגר cx יכיל את המיקום האופקי (X) ואוגר dx יכיל את המיקום האנכי (Y).

```
PutPixel ( int x , int y , byte color )
{
  union REGS regs ;

  regs.h.ah = 0x0c ;
  regs.h.al = color ;
  regs.x.cx = x ;
  regs.x.dx = y ;

  int86 ( 0x10 , &regs , &regs ) ;
}
```

שיטה זו לשתילת פיקסלים אמנם פשוטה, אך איטית בגלל השימוש בפסיקת BIOS.

בעבודה בגרפיקה, ובמיוחד כשרוצים לשלב אנימציה, מהירות הביצוע היא קריטית. על כן נשאף תמיד להשתמש בדרך המהירה ביותר לציור פיקסלים על המסך. שיטה מהירה יותר לשתילת פיקסל על המסך היא על ידי גישה ישירה לזיכרון.

חוצץ הזיכרון המוקצה לנתוני המסך במוד 13H מתחיל מכתובת A000H וגודלו 64KB, לכן נגדיר מצביע על תחילת החוצץ.

```
unsigned char byte ;
byte *VGA = (byte *) MK_FP (0xA000,0) ;
```

מכיון שהזיכרון הוא במבנה ליניארי ולא במבנה מטריצה כמו המסך, כל 320 בתים מייצגים שורה במסך. 320 הבתים הראשונים בחוצץ מוקצים לשורה העליונה במסך, 320 הבתים הבאים מייצגים את השורה השנייה, וכך הלאה. על כן צריך לחשב מהו ההיסט (offset) המתאים לקואורדינטה המבוקשת.

לשם כך יש לכפול את מספר השורה (y) ב-320 ולהוסיף את מספר הטור (x).

```
VGA [ 320*y + x ] = color ;
```

שיטה ישירה זו מהירה בהרבה מהשיטה הקודמת, אך גם אותה ניתן להפוך למהירה יותר על ידי שימוש בהזזת סיביות. כפי שלמדנו בפרק 2, "עבודה בסיביות", משך הזמן הדרוש למחשב לביצוע פעולת כפל גדול, לכן ניתן להשתמש בהזזת סיביות כדי לשפר את מהירות הביצוע.

אך כאן יש לנו בעיה! בשיטת הכפל באמצעות הזזת סיביות, ניתן לכפול רק בחזקות של שתיים, והמספר 320 אינו חזקה של 2. על כן, נפרק את המספר לשני חלקים, 256 ו-64, שהם חזקות של 2 וסכומם הוא 320 (2 בחזקת 8 + 2 בחזקת 6 = 320). כעת נוכל לכתוב כך את הפונקציה לשתילת הפיקסל:

```
PutPixel ( int x , int y , byte color )
{
  VGA [   ( y<<8 ) + ( y<<6 ) + x  ] = color ;
}
```

נראה כאילו שינויים אלה זניחים ולא חשובים, אך כשתכתבו תוכניות, תוכלו לראות עד כמה ההבדל משמעותי, כי תצטרכו לשתול מאות אלפי פיקסלים בזמן קצר מאוד.

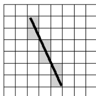


נסו לכתוב תוכנית המשווה בין שתי השיטות, ובודקת פי כמה הגישה הישירה מהירה משיטת הפסיקות. בצעו זאת על ידי שתילת 5000 פיקסלים בכל אחת מהשיטות.

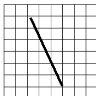
9.4 ציור קווים

יצירת קו ישר אופקי או אנכי, יכולה להתבצע בלולאה פשוטה המקדמת את המשתנה x או את המשתנה y בהתאמה. אך מה עושים כשהקו הוא אלכסוני בעל זווית כלשהי?

אם ברצוננו לצייר על המסך קו, כמו הקו שמוצג בתרשים 9.1, נצטרך לצבוע את הפיקסלים שהוא עובר דרכם (כמו בתרשים 9.2). כדי למצוא את הפיקסלים האלה נצטרך להשתמש בנוסחאות של הנדסה אנליטית לחישוב שיפוע ומרחק.



תרשים 9.1



תרשים 9.2

באלגוריתם לציור קו, נצטרך לדעת מהו הסימן של משתנה כלשהו (האם הוא שלילי, חיובי או אפס). לשם כך ניצור פונקציה כגון זו:

```
int sgn ( int num )
{
  if ( num>0 ) return ( 1 ) ;
  else if ( num<0 ) return ( -1 ) ;
         else return ( 0 ) ;
}
```

לחילופין, נוכל לעשות זאת בצורה יותר אלגנטית ולכתוב מאקרו בראש התוכנית.

```
#define sgn(x) ( (x<0) ?-1 : ( (x>0) ?1 :0 ) )
```

להלן האלגוריתם המוכר לציור קו. האלגוריתם מורכב, עקב השימוש בנוסחאות מתמטיות. אין צורך להתעמק בו, אך חשוב להכירו ולהבינו.

```
void DrawLine (int x1, int y1, int x2, int y2, byte color)
/* פונקציה המציירת קו בין הנקודות (x1,y1) ו- (x2,y2) */
{
   int i , dx , dy, sdx , sdy , x , y , px , py ;
  /* המשתנים מוגדרים כשלמים למרות שתוצאות החישובים אינן שלמות
                                    וכך התוצאות יתעגלו באופן אוטומטי */
```

```
dx = x2-x1;              // המרחק האופקי בין הנקודות
dy = y2-y1;              // המרחק האנכי בין הנקודות

sdx= sgn(dx);            // הסימן
sdy= sgn(dy);            // הסימן

dx = abs(dx);            // ערך מוחלט
dy = abs(dy);            // ערך מוחלט

x  = dy>>1;              // מקבל את מחצית הערך
y  = dx>>1;              // מקבל את מחצית הערך

px=x1;  py=y1;           // מיקום הנקודה הראשונה
PutPixel(px,py,color);   // שתילת הנקודה הראשונה

if (dx>=dy)              // אם הקו יותר אופקי מאנכי
{
   for(i=0;i<dx;i++)
   {
      y+=dy;
      if (y>=dx)
      {   y-=dx ;     py+=sdy;  }
      px+=sdx ;     PutPixel(px,py,color,Page);
   }
}
else                     // אם הקו יותר אנכי מאופקי
{
   for(i=0;i<dy;i++)
   {
      x+=dx;
      if (x>=dy)
      {   x-=dy ;    px+=sdx;    }
      py+=sdy ;    PutPixel(px,py,color,Page) ;
   }
}
```



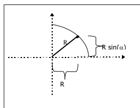
השתמשו בפונקציות שלמדתם וכתבו פונקציות המציירות מלבן חלול, מלבן מלא, משולש ופונקציה הצובעת את כל המסך.

9.5 ציור עיגולים

כדי לצייר עיגול, משתמשים בטריגונומטריה של המעגל. על פי חוקי הטריגונומטריה, כשנתונה זווית α, מיקום נקודת הגיזרה על היקף המעגל יהיה: $(R \cos(\alpha), R \sin(\alpha))$, (כפי שתראו בתרשים 9.3).

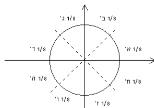
כמו כן ידוע כי: $\alpha = \arccos(x/R)$

ולכן ניתן לומר כי: $y = R \sin(\arccos(x/R))$



תרשים 9.3

באמצעות נוסחאות אלו, נוכל לדעת את המיקום של כל הנקודות על היקף המעגל, כאשר נתונים נקודת המרכז והרדיוס שלו. אך במקום לבצע חישובים רבים, ניתן לחלק את המעגל לשמונה חלקים (שמיניות), לצייר שמינית אחת ולשכפל אותה כדי לקבל עיגול שלם (ראו תרשים 9.4).



תרשים 9.4

```
void DrawCircle ( int x , int y , int R , byte color )
{
typedef unsigned short word ;
  float n=0 , invr = 1/(float)R ;
  int   dx=0 , dy=R-1;
  word dxoffset , dyoffset , offset= (y<<8) + (y<<6) + x ;

  while (dx<=dy)
  {
    dxoffset = (dx<<8) + (dx<<6) ;
    dyoffset = (dy<<8) + (dy<<6) ;

    VGA [ offset + dy - dxoffset ] = color ;   // שמינית א'
    VGA [ offset + dx - dyoffset ] = color ;   // שמינית ב'
    VGA [ offset - dx - dyoffset ] = color ;   // שמינית ג'
    VGA [ offset - dy - dxoffset ] = color ;   // שמינית ד'
    VGA [ offset - dy + dxoffset ] = color ;   // שמינית ה'
    VGA [ offset - dx + dyoffset ] = color ;   // שמינית ו'
    VGA [ offset + dx + dyoffset ] = color ;   // שמינית ז'
    VGA [ offset + dy + dxoffset ] = color ;   // שמינית ח'
    dx ++ ;
    n += invr ;
    dy = R * sin (acos (n) ) ;
  }
}
```



לא לשכוח שיש לקרוא ל- Math.h בראש התוכנית לשם ביצוע פעולות טריגונומטריה.

אם התוכנית שברצונכם לכתוב, צריכה לצייר מספר גדול של עיגולים, רצוי להכין טבלה שתכיל את תוצאות החישובים הטריגונומטריים, ולא לחשבם מחדש בנפרד לכל עיגול.



כתבו פונקציה המציירת עיגול מלא (לא מעגל חלול).

9.6 לוח הצבעים

כפי שהסברנו בתחילת הפרק, מוד 13H תומך ב-256 צבעים, אשר ממוספרים מאפס עד 255. חשוב להבין שצבעים אלה אינם קבועים, והבחירה לגבי הגוונים שיופיעו בלוח הצבעים (color palette), היא בידי המתכנת.

כיצד יוצרים גוונים שונים?

כידוע, כל צבע מורכב משלושת צבעי היסוד (אדום, ירוק וכחול, RGB). אם נשנה את דרגת החוזק של אחד מהם, ישתנה הצבע הסופי.

במחשב, דרגת החוזק של צבע יסוד מיוצגת על ידי ערך בין אפס ל-63. כל צבע מתוך 256 הצבעים שבלוח הצבעים מיוצג על ידי 3 ערכים (בגודל בית כל אחד), המסמלים את רמת החוזק של כל אחד משלושת צבעי היסוד.

למעשה באפשרותנו ליצור כ- 250,000 גוונים שונים! אך בו-זמנית נוכל להציג רק 256 גוונים שונים.

כיצד שולפים את ערכי צבעי היסוד שמהם בנוי צבע מסוים?

תחילה יש לשלוח לפורט (יציאה) 3c7H את מספר הצבע המבוקש.

לאחר מכן, יש לקרוא מפורט 3c9H את שלושת הערכים של צבעי הבסיס לפי הסדר הזה: אדום, ירוק, כחול.

```
void GetPal(byte ColorNo , byte *R , byte *G , byte *B)
{
  outp (0x03C7,ColorNo);        // שליחת מספר הצבע המבוקש
  *R = inp (0x03C9);            // שליפת עוצמת הצבע האדום
  *G = inp (0x03C9);            // שליפת עוצמת הצבע הירוק
  *B = inp (0x03C9);            // שליפת עוצמת הצבע הכחול
}
```

כדי להגדיר ערכים חדשים לצבע מסוים בלוח הצבעים, יש לשלוח לפורט 3c8H את מספר הצבע המבוקש. אחר כך יש לשלוח לפורט 3c9H את שלושת הערכים של צבעי הבסיס לפי סדר זה: אדום, ירוק, כחול.

```
void SetPal(byte ColorNo , byte R , byte G , byte B)
{
  outp (0x03C8,ColorNo);        // שליחת מספר הצבע המבוקש
  outp (0x03C9,R);              // שליחת עוצמת הצבע האדום
  outp (0x03C9,G);              // שליחת עוצמת הצבע הירוק
  outp (0x03C9,B);              // שליחת עוצמת הצבע הכחול
}
```

לדוגמה, אם נכניס לתא של צבע מספר 135 את הערכים שבתא צבע מספר 12, ונצייר על המסך קו בצבע 135 וקו בצבע 12 – הם ייראו לנו אותו הדבר.

לפני שעורכים שינויים בלוח הצבעים, רצוי לשמור את הערכים הנמצאים בו. לשם כך יש להגדיר מטריצה גלובלית:

```
byte Pall [256][3] ;
```

כעת נכתוב פונקציה הקוראת את כל ערכי הצבעים שבלוח הצבעים (palette) לתוך המטריצה שבנינו:

```
void GrabPalette(void)
{
  int i ;
  for(i=0;i<256;i++)
    GetPal(i,&Pall[i][0],&Pall[i][1],&Pall[i][2]);
}
```



1. כתבו פונקציה שתמלא את כל הלוח בגוונים שונים של כחול.
2. כתבו פונקציה המסתירה את כל מה שמצויר על המסך.

Fade Out / Fade In 9.7

אפקט מרשים ופשוט, שניתן לעשות בעזרת לוח הצבעים הוא Fade In ו-Fade Out (מעמעם, מגלה באיטיות את כל הצבעים במסך).

כיצד עושים Fade Out?

Fade Out מכהה בהדרגה את כל הצבעים במסך, עד שכל המסך הופך לשחור. כדי לעשות זאת, עלינו להוריד באיטיות את העוצמה של צבעי היסוד בכל אחד מהתאים בלוח, עד שכולם יהיו אפס (צבע שחור).

```
void FadeOut(void)
{
  byte ColBase [3];              // מערך של שלושה בתים המייצגים צבע
  int I,j ;

  for(I=0; I<64; I++)            // עד 64 כי זהו החוזק המקסימלי
  {
    delay(50) ;                  // השהיה
    for(j=0 ; j<256; j++)        // כמספר הצבעים בלוח הצבעים
    {
                                 // קרא את ערכי הצבע  לתוך המערך
        GetPal(j,&ColBase[0],&ColBase[1],&ColBase[2]) ;
```

```
                    // אם צבע הבסיס גדול מאפס, הורד ממנו אחד.
         if (ColBase[0] > 0) ColBase[0]--;
             if (ColBase[1] > 0) ColBase[1]--;
             if (ColBase[2] > 0) ColBase[2]--;

                    // קבע את ערכי הצבע המתוקנים.
        SetPal(j,ColBase[0],ColBase[1],ColBase[2]);
    }
  }
}
```

כיצד עושים Fade In?

פעולת Fade In הפוכה לפעולת Fade Out: מתחילים ממצב של מסך חשוך ובהדרגה מבהירים את הצבעים, עד שהם תואמים את צבעי הלוח המקוריים (לשם כך חשוב לזכור להפעיל בתחילת התוכנית את הפונקציה ()GrabPalette).

```
void FadeIn (void)
{
  byte ColBase [3];              // מערך של שלושה בתים המייצגים צבע
  int I,j ;

  for(I=0; I<64; I++)            // עד 64 כי זהו החוזק המקסימלי
  {
    delay(30) ;                  // השהיה
    for(j=0 ; j<256; j++)        // כמספר הצבעים בלוח הצבעים
    {
                                 // קרא את ערכי הצבע לתוך המערך
    GetPal(j,&ColBase[0],&ColBase[1],&ColBase[2]) ;

            // לכל צבע בסיס, אם הוא קטן ממה שהוא צריך להיות, קדמו אותו באחד.
    if ((ColBase[0] < Pall[j][0]) && (ColBase[0] < 63))
      ColBase[0]++;
    if ((ColBase[1] < Pall[j][1]) && (ColBase[1] < 63))
      ColBase[1]++;
    if ((ColBase[2] < Pall[j][2]) && (ColBase[2] < 63))
      ColBase[2]++;

                              // קבע את ערכי הצבע המתוקנים
      SetPal(j,ColBase[0],ColBase[1],ColBase[2]);
    }
  }
}
```

לאחר הרצת התוכנית, ניתן לראות, שישנם מדי פעם עיוותים קטנים בנקודות שונות בזמן ביצוע Fade In או Fade Out. מדוע זה קורה?

בתוך המסך יש תותח אלקטרונים, המעדכן בקביעות את הנתונים המוצגים במרקע. העיוותים נוצרים, מכיון שהלוח משתנה בזמן שהתותח צובע את המרקע.

כיצד לפתור את בעיית העיוותים?

בין פעולות צביעת המרקע, תותח האלקטרונים נמצא בהשהיה הנקראת **Vertical Retrace**. אם נדאג שהשינויים בלוח יתבצעו בפרק הזמן של השהיית התותח, בסוף ההשהיה, התותח יצבע מחדש את המסך עם ערכי הלוח המחודשים וכך נוכל למנוע את העיוותים.

כדי לדעת מתי תותח האלקטרונים נמצא בהשהיה, יש להשתמש בפונקציה זו:

```
void WaitRetrace ()
{
  byte al ;

            //    לולאה הקוראת מפורט 3DAH ערך, ובודקת את הסיבית הרביעית
            //    אם ערכה הוא אחד לוגי, התותח נמצא באמצע ההשהיה ולכן חכה
            //    עד שההשהיה תיגמר (אנחנו רוצים למצוא את תחילת ההשהיה).
  do
  {
    al = inp (0x03DA) ;
    al &= 0x08 ;
  } while (al) ;


            //    לולאה הקוראת מפורט 3DAH ערך, ובודקת את הסיבית הרביעית
            //    אם ערכה הוא אפס לוגי , התותח נמצא באמצע הצביעה ולכן חכה
            //    עד שהצביעה תיגמר כי אז תתחיל ההשהיה.
do
  {
    al = inp (0x03DA) ;
    al &= 0x08 ;
  } while (!al) ;
}
```

עכשיו, כל מה שנותר לעשות זה, להוסיף קריאה לפונקציה הזאת בתוך פונקציית העמעום (FadeOut), ובפונקציית ההתבהרות (FadeIn), הקריאה צריכה להיות בסמוך לפקודה Delay (השהיה). כמו כן, מומלץ להקטין את משך ההשהיה משום שהפונקציה ()WaitRetrace מאיטה בעצמה את פעולת ה-Fade.

9.8 דפים וירטואליים

מהו דף וירטואלי?

דף וירטואלי הוא קטע זיכרון הזהה בגודלו לחוצץ של המסך שעמו אנו עובדים (במקרה שלנו – 64KB). במקום לצייר ישירות על המסך, ניתן לצייר על דף וירטואלי ואחר כך להעתיק את כל תוכן הדף ישירות אל חוצץ המסך.

מדוע רצוי להשתמש בדפים וירטואליים?

יש מספר יתרונות לשימוש בדפים וירטואליים:

1. השימוש בדפים וירטואליים מהיר יותר מציור ישיר על המסך.
2. כשמשתמשים בדפים וירטואליים לא רואים איך הציור נוצר על המסך.
3. כאשר רוצים להניע ציור כלשהו על המסך, לדוגמה מלבן, צריכים לצייר את המלבן במקום החדש ואחר כך למחוק את המלבן הקודם. אם נעשה זאת ישירות על המסך תיווצר הכפלה של הציור, כי עד שיימחק המלבן הקודם יוצגו לפנינו שני מלבנים.
4. אם לדוגמה, מאחורי המלבן שברצוננו להזיז מצויר משהו אחר, לא נוכל למחוק את המלבן ונצטרך לצייר הכל מחדש. בשיטת הדפים הווירטואליים, נוכל לשמור את הרקע בדף וירטואלי ואז בכל תזוזה של המלבן לטעון אותו מחדש בלי המלבן, ועליו נלביש את המלבן במיקום החדש.

השימושים בדפים וירטואליים הם רבים ומגוונים, וכדאי שנעבוד בשיטה זו. כדי להשתמש בדפים וירטואליים יש להקצות להם מקום בזיכרון. מספר הדפים הווירטואליים שמוקצים לעבודה נתון לשיקול המתכנת, אך חשוב לזכור שכל דף גוזל מקום יקר בזיכרון.

אם אין יותר מדי פעלולים בתוכנית, שני דפים וירטואליים יספיקו בהחלט.

תחילה יש להגדיר בראש התוכנית מצביעים גלובליים שיצביעו על האזור בזיכרון שמוקצה לדפים הווירטואליים.

```
byte *VPage1 = NULL ;
byte *VPage2 = NULL ;
```

עכשיו יש להקצות את המקום בזיכרון לדפים הווירטואליים, לשם כך נכתוב פונקציה זו:

```
void OpenVPage (void)
{
  VPage1 = (byte *) calloc(64000,1) ;  // הקצאת הזיכרון לדף 1
  VPage2 = (byte *) calloc(64000,1) ;  // הקצאת הזיכרון לדף 2
```

```
if ( VPage1==NULL || VPage2==NULL )  // אם ההקצאה נכשלה
{
   SetTextMode ;
   printf ("אין מספיק זיכרון") ;
   exit (1) ;
}
}
```

אם לא נשחרר את המקום שהקצנו לדפים הווירטואליים בסוף השימוש בהם, הוא יישאר חסום לאחר היציאה מהתוכנית. על כן יש לכתוב פונקציית שחרור ולהפעילה בתום התוכנית.

```
void CloseVPage (void)

{
  free (VPage1) ;
  free (VPage2) ;
}
```

לאחר יצירת הדפים הווירטואליים, אנו יכולים להשתמש בהם. לשם כך עלינו להוסיף משתנה מצביע מסוג בית לכל אחת מפונקציות הציור. במקום לחשב את ההיסט מתחילת חוצץ VGA, נחשב אותו מהמצביע המועבר. נשמע מסובך, אך למעשה זה פשוט מאוד.

ניקח לדוגמה את הפונקציה שכתבנו לשתילת פיקסל על המסך:

```
PutPixel ( int x , int y , byte color )
{
  VGA [   ( y<<8 ) + ( y<<6 ) + x  ] = color ;
}
```

כל שעלינו לעשות הוא להוסיף לרשימת הפרמטרים את המצביע Page, ובמקום לחשב את ההיסט מהמקום עליו מצביע המצביע VGA, נחשב אותו מהמקום שמצביע Page מורה עליו.

הפונקציה המתוקנת תיראה כך:

```
void PutPixel (int x, int y, byte color, byte *Page)
{
  Page [(y<<8)+(y<<6)+x] = color ;
}
```

עכשיו, אם נכתוב את הפקודה:

```
PutPixel (13,14,100,Vpage1) ;
```

נשתול למעשה פיקסל בדף וירטואלי מספר 1.

ואם נכתוב את הפקודה:

```
PutPixel (13,14,100,VGA) ;
```

נשתול פיקסל ישירות על המסך.

נכון שהשד לא כל כך נורא?

בצורה זו יש לתקן את כל פונקציות הציור שכתבנו (קו, עיגול, מלבן וכו').

עכשיו, כל מה שנותר הוא לכתוב פונקציה הטוענת תוכן של דף וירטואלי אל המסך ולהיפך. נעשה זאת על ידי העתקת הזיכרון בצורה זו:

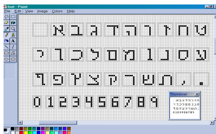
```
void LoadPage (byte *Page1 , byte *Page2)
{
  _fmemcpy(Page1,Page2,64000) ;
}
```

9.9 יצירת גופנים

הנושא האחרון לפרק זה הוא יצירת גופנים. מכיון שאנו פועלים במוד 13H, עלינו ליצור בעצמנו גופנים כדי שנוכל לשלב טקסט בתוכנית שנכתוב.

כיצד יוצרים גופנים?

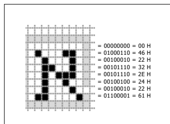
כדי ליצור גופנים, מומלץ להיעזר בתוכנת הצייר של Windows, לעשות זום מקסימלי ולהוסיף קווי רשת (ראו תרשים 9.5). מומלץ לייצג כל אות ב-64 משבצות (8*8), כי כל בית מכיל 8 סיביות.



תרשים 9.5

עכשיו אפשר להפעיל את הדימיון, לצבוע משבצות בשחור ולצייר את האותיות הרצויות. לאחר ציור כל האותיות, עלינו לייצג כל שורה של האות על ידי מספר הקסדצימלי.

ניקח לדוגמה את האות 'אלף' ונתייחס אל המשבצות כאל סיביות בינאריות. משבצת לבנה "מייצגת 'אפס' ומשבצת שחורה מייצגת 'אחת'. עכשיו עלינו להמיר את המספרים הבינאריים למספרים הקסדצימליים. מספר בינארי מיוצג על ידי 8 סיביות וכאשר ממירים אותו לייצוג הקסדצימלי מקבלים שתי ספרות (כי כל 4 סיביות מייצגות תו הקסדצימלי, כפי שלמדנו בפרק 2, "עבודה בסיביות"). כך נייצג כל אות על ידי 8 מספרים הקסדצימליים (אחד לכל שורה).



תרשים 9.6

אתם ודאי חושבים עכשיו – מהו השיגעון הזה, למי יש כוח לעשות זאת, ולמה אי אפשר רק לשמור את התמונה ולטעון אותה למסך? ובכן, בפרק הבא נלמד לטעון תמונות מסוג BMP, אך במקרה זה עדיף לבצע את השיטה הארוכה מכמה סיבות:

1. בשיטה זו כל נקודה מיוצגת על ידי סיבית אחת ולא על ידי בית. בקובץ BMP יש כותר התופס בתים רבים (פירוט בפרק הבא), ולכן קובץ הגופן בשיטה שלנו יהיה קטן בהרבה!
2. מהירות כתיבת האותיות בשיטה זו הרבה יותר גבוהה.
3. בזכות צורת ייצוג זו נוכל בקלות ליצור אפקטים מרשימים בסגנון WordArt.
4. לאחר שמסיימים עבודה שחורה זו פעם אחת, לפנינו קובץ גופן מוכן לשימוש.
5. ממילא כבר יש לפניכם חלק ניכר מהעבודה (קובץ הגופנים העבריים וקובץ הספרות מצורפים בדיסק).

אני מקווה שהשתכנעתם שכדאי בכל זאת להשקיע את המאמץ.

לפני שאמשיך בהסבר, יש עוד כמה הערות :

1. שימו לב שאות סופית מוצגת לפני האות הרגילה. הסיבה לכך היא שזהו סדר אותיות העברית בטבלת ASCII. הסדר חשוב, כי הוא יקל על מציאת האות.
2. את הספרות בחרתי לייצג במטריצה 7*4, כי הספרות יותר צרות מהאותיות. לכן יצרתי 2 קבצים (המצורפים בדיסק) :

   1 – Digits.mft (מכיל את גופני הספרות).

   2 – Letters.mft (מכיל את גופני האותיות).

לאחר סיום כל החישובים (המרת הציור לערכים הקסדצימליים), יש לשמור את כל הערכים בקובץ.

להלן התוכנית שבעזרתה שמרתי את ערכי הספרות בקובץ Digits.mft :

```
#include <stdio.h>
#include <conio.h>

typedef unsigned char byte ;

void main (void)
{
byteDigits[10][7]={
{0x6,0x9,0x9,0x9,0x9,0x9,0x6},{0x2,0x6,0x2,0x2,0x2,0x2,0x7},
{0xE,0x1,0x1,0xF,0x8,0x8,0xF},{0xE,0x1,0x1,0x7,0x1,0x1,0xE},
{0x9,0x9,0x9,0xF,0x1,0x1,0x1},{0xF,0x8,0x8,0xF,0x1,0x1,0xF},
{0xF,0x8,0x8,0xF,0x9,0x9,0xF},{0xF,0x1,0x2,0x2,0x4,0x4,0x8},
{0xF,0x9,0x9,0xF,0x9,0x9,0xF},{0x6,0x9,0x9,0x7,0x1,0x1,0x1} } ;

  FILE *pf ;
  pf = fopen ("c:\\digits.mft","wb") ;
  fwrite(Digits,sizeof(byte),70,pf) ;
  close(pf) ;
}
```

בשיטה דומה שמרתי גם את ערכי האותיות בעברית. כך גם אפשר לשמור את ערכי האותיות באנגלית.

עכשיו הגיע זמנכם לעבוד קצת . . .



עצבו את האותיות באנגלית כמו שעיצבתי את האותיות בעברית.
הוסיפו את ערכי האותיות באנגלית אל קובץ האותיות.

כיצד משלבים את האותיות שבנינו בתוך התוכנית?

תחילה יש להגדיר שתי מטריצות גלובליות בראש התוכנית ולטעון לתוכן את ערכי האותיות והספרות.

```
byte Digits [10][7] ;  // מטריצה לערכי הספרות
byte Letters[30][8] ;  // מטריצה לערכי האותיות
```

כדי לטעון את ערכי הגופנים מתוך הקבצים אל המטריצה נשתמש בפונקציות שונות, לדוגמה:

```
void LoadDigits ()

{
  FILE *pf ;
  pf = fopen ("c:\\digits.mft","rb") ;
  fread(Digits,sizeof(byte),70,pf) ;
  close(pf) ;
}
```

וכך גם לאותיות...

עכשיו עלינו להציג את הספרה או האות על המרקע. כדי לדעת אם ערך הסיבית הוא 'אפס' או 'אחת', נצטרך להשתמש בפונקציה לבדיקת סיבית, שהכרנו בפרק 2.

```
int TestBit (byte X, int pos)
{
  X >>= pos ;
  return ( X &= 0x1 ) ;
}
```

כעת צריך להכניס את בדיקת הסיבית ללולאה מקוננת, שתרוץ על כל אחת מהסיביות המרכיבות את האות. אם ערך הסיבית הוא 'אחת' יישתל פיקסל על המסך.

```
void PutLetter (char let, int x, int y, byte color,
                  byte *Page) {
  int i , j , pos ;

  // מציאת האות במטריצה
  if (let >= 'א' && let <= 'ת' )
    pos = let - 'א'+ 1 ;
  else if (let == ',')
    pos = 28 ;
  else if (let == '.')
    pos = 29 ;
  else pos = 0  ;
```

```
  for (i=0;i<8;i++)           //  לכל שורה באות
  {
    for(j=0;j<8;j++)          // לכל טור באות
    {
      if (TestBit (Letters[pos][i],j))
        PutPixel (x-j,y+i,color,Page) ;
} } }
```

ניתן לנצל את העובדה שהאותיות מצוירות סיבית אחר סיבית, כדי ליצור אפקטים מרהיבים. לדוגמה, אם נקדם באחת את הצבע בכל טור באות, תיווצר אות הצבועה בדירוג. אם לכל אות נשלח צבע התחלתי גדול באחת, המילה תיראה כאילו היא צבועה בדירוג אלכסוני (ראו תרשים 9.7).

1234567890

תרשים 9.7

```
void PutDigit ( int dig ,int x , int y, byte color ,byte
          fx,byte *Page)
{
  int i , j ;
  for (i=0;i<7;i++)          // לכל שורה בספרה
  {
    for(j=0;j<4;j++)         // לכל טור בספרה
    {
      if (TestBit (Digits[dig][i],j))
        PutPixel (x-j,y+i,color,Page) ;
    }
    if (fx) color++ ;        // אם האפקט מופעל קדם את הצבע
  }
}
```

ואת האפקט נוסיף גם לפונקציה הקודמת . . .

לסיום, צריך לאחד את שתי הפונקציות לפונקציה אחת המקבלת מחרוזת, מיקום וצבע התחלתי, ומדפיסה אותה על המסך.

```
void PutText (char st[],int x,int y,byte color,
                    byte fx,byte *Page) {
  int p;

  for(p=strlen(st)-1;p>=0;p--) // עבור כל אות במחרוזת מימין לשמאל
  {
    if (st[p] >= '0' && st[p] <= '9')  // אם זוהי ספרה
    {
      PutDigit (st[p]-'0',x,y+1,color,fx,Page );
      x -= 6 ;
    }
    else        // אם זוהי אות
    {
      PutLetter (st[p],x,y,color,fx,Page ) ;
      x -= 9 ;
    }

    if (fx) color ++ ;
  }
}
```

9.10 חלונות

בתוכנות מחשב רבות, המידע מוצג על המסך בחלונות המצויירים זה על זה (כמו במערכת ההפעלה window). נסו לחשוב כיצד לממש את התצוגה המדורגת של חלונות אלה.

הדרך היעילה ביותר לעשות זאת, היא בעזרת **מחסנית** הפועלת בשיטת last in first) LIFO out), שבה כל איבר במחסנית מייצג חלון על המסך. החלון העליון ביותר במסך הוא החלון שנמצא בראש המחסנית והתחתון ביותר – בסוף המחסנית. אם המשתמש לוחץ בעכבר על חלון שאינו החלון העליון ביותר, נוציא את האיבר המתאים מהמחסנית ונדחוף אותו לראש המחסנית.

אם אינכם יודעים כיצד לממש מחסנית בשפת C, תוכלו לפנות לספר "**שפת C - תוכניות ופתרונות**", בהוצאת הוד-עמי.

מהו המידע שיש לשמור עבור כל חלון בתוך המחסנית? כל איבר במחסנית צריך להכיל מידע אודות מיקום החלון, גודל החלון ותוכנו. אם התוכנה צריכה לתמוך במספר רב של חלונות ונשמור עבור כל חלון את תמונתו, נגזול מקום רב בזיכרון. לכן, עדיף לשמור על כל חלון רק את ההוראות כיצד לצייר את תוכנו. כך נוכל לחסוך בזיכרון, כי אין צורך לשמור מידע על חללים ריקים בחלון, אלא צריך לשמור את ההנחיות לצייר אותו בלבד.

נסו לחשוב מה קורה כשהמשתמש מנסה להזיז חלון במסך. מהן הבעיות שעלולות להיווצר וכיצד נתגבר עליהן?

ברגע שהמשתמש מזיז חלון במסך, התוכנה צריכה לעדכן את נתוני מיקום החלון במחסנית ובמקביל עליה לעדכן גם את התצוגה במסך. אם עדכון המסך היה מתבצע על פי המחסנית, התוכנית היתה צריכה לצייר מחדש לא רק את החלון שאותו המשתמש מזיז, אלא את כל החלונות על המסך. שיטה זו אינה יעילה ואם ננהג כך, נראה שיבושים בתצוגה, כי התוכנית לא מספיקה לצייר את כל החלונות כתגובה לכל תזוזה קטנה שהמשתמש מבצע.

כדי להתגבר על בעיה זו נשתמש בדפים וירטואליים, כפי שלמדנו בסעיף 9.8. נשמור בדף וירטואלי את תוכן המסך ללא החלון המסוים שהמשתמש מנסה להזיז. בכל תזוזה של החלון נעתיק אל חוצץ המסך את תוכן הדף הווירטואלי, ונצייר במקום החדש, את החלון שהוזז. שיטה זו הרבה יותר יעילה, כי בכל תזוזה התוכנית צריכה לצייר תוכן של חלון אחד בלבד, ואינה צריכה לצייר את כל החלונות. כדי למנוע שיבושים בתצוגה, נצייר את החלון בזמן השהיית תותח האלקטרונים של המסך, כפי שלמדנו בסעיף 9.7.

9.11 יחידה לגרפיקה ב-256 צבעים

ממשק היחידה

```
/*************************************************************************
 *                           V G A 2 5 6 . H                             *
 *                       ~~~~~~~~~~~~~~~~~~~~~~                          *
 *            ממשק יחידה לטיפול במסך גרפי באיכות 256 צבעים              *
 *************************************************************************/

//////////////////////////////////////////////////////////////////////////
//                          ה  צ  ה  ר  ו  ת                            //
//////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <dos.h>
#include <mem.h>
#include <math.h>

#define SCREEN_WIDTH  320
#define SCREEN_HEIGHT 200

#define sgn(x)  ((x<0)?-1:((x>0)?1:0)) // מאקרו שמחזיר את סימנו של מספר (0,-1,1)
```

```
typedef unsigned char  byte;
typedef unsigned short word;
typedef unsigned long  dword;

///////////////////////////////////////////////////////////////////////////
//                         פ ו נ ק צ י ו ת                              //
///////////////////////////////////////////////////////////////////////////

void Set256Mode (void);
/*-------------------------------------------------------------------------
                                  פונקציה להפעלת מוד 13H (56 צבעים).
                                             פרמטרים מועברים - אין.
                                                   ערך מוחזר - אין.
-------------------------------------------------------------------------*/

void SetTextMode (void);
/*-------------------------------------------------------------------------
                                        פונקציה להפעלת מוד טקסט רגיל
                                              פרמטרים מועברים - אין
                                                    ערך מוחזר - אין
-------------------------------------------------------------------------*/

void OpenVPage (void) ;
/*-------------------------------------------------------------------------
                               פונקציה המקצה מקום לדפים הווירטואליים
                                              פרמטרים מועברים - אין
                                                    ערך מוחזר - אין
-------------------------------------------------------------------------*/

void CloseVPage (void) ;
/*-------------------------------------------------------------------------
                 פונקציה המשחררת את הזיכרון שהוקצה לדפים הווירטואליים
                                              פרמטרים מועברים - אין
                                                    ערך מוחזר - אין
-------------------------------------------------------------------------*/

void WaitRetrace ();
/*-------------------------------------------------------------------------
                 פונקציה המחכה שתותח האלקטרונים יהיה בתחילת השהיה
                                              פרמטרים מועברים - אין
                                                    ערך מוחזר - אין
-------------------------------------------------------------------------*/
```

```
void PutPixel (int x, int y, byte color , byte *Page) ;
/* --------------------------------------------------------------------------
פונקציה השותלת פיקסל בהיסט המחושב על פי x ו- y מתחילת המצביע Page
פרמטרים מועברים
- x -      מיקום אופקי.
- y -      מיקום אנכי.
- color - צבע.
- page  - היכן לצייר (מסך או דף וירטואלי).
ערך מוחזר - אין
--------------------------------------------------------------------------*/

void DrawLine (int x1, int y1, int x2, int y2, byte color, byte *Page);
/*---------------------------------------------------------------------------
פונקציה לציור קו על דף/מסך
פרמטרים מועברים :
- x1,y1 -  קואורדינטות הנקודה השמאלית עליונה.
- x2,y2 -  קואורדינטות הנקודה הימנית תחתונה.
- color - הצבע.
-  Page - היכן לצייר (מסך או וירטואלי ).
ערך מוחזר - אין.
--------------------------------------------------------------------------*/

void DrawEmptyRect (int x1, int y1, int x2, int y2, byte color, byte *Page);
/*---------------------------------------------------------------------------
פונקציה לציור מלבן חלול על דף/מסך.
פרמטרים מועברים :
- x1,y1 -  קואורדינאטות הנקודה השמאלית עליונה.
- x2,y2  -  קואורדינאטות הנקודה הימנית תחתונה.
- color  -   הצבע
- page  -   היכן לצייר (מסך או וירטואלי)
ערך מוחזר - אין.
--------------------------------------------------------------------------*/

void DrawFilledRect (int x1,int y1,int x2,int y2, byte color, byte *Page);
/*---------------------------------------------------------------------------
פונקציה לציור מלבן מלא על דף/מסך.
פרמטרים מועברים :
- x1,y1 -  קואורדינטות הנקודה השמאלית עליונה.
- x2,y2 -  קואורדינטות הנקודה הימנית תחתונה.
- color - הצבע.
- Page - היכן לצייר ( מסך או וירטואלי ).
ערך מוחזר - אין.
--------------------------------------------------------------------------*/
```

```
void ScreenColor (byte color, byte *Page);
/*--------------------------------------------------------------------------
        פונקציה הצובעת את הדף/מסך.
        פרמטרים מועברים :
            - color - הצבע.
            - Page - היכן לצייר ( מסך או וירטואלי ).
        ערך מוחזר - אין.
--------------------------------------------------------------------------*/

int TestBit (byte X, int pos);
/* -------------------------------------------------------------------------
        פונקציה הבודקת האם סיבית מסוימת בבית היא במצב לוגי "אמת".
        פרמטרים מועברים :
            - X - הבית לבדיקה.
            - pos - איזו סיבית לבדוק.
        ערך מוחזר :  מצב הסיבית
--------------------------------------------------------------------------*/

void LoadDigits ();
/*--------------------------------------------------------------------------
        פונקציה הטוענת את הגופנים של הספרות למטריצה.
        פרמטרים מועברים - אין.
        ערך מוחזר - אין.
--------------------------------------------------------------------------*/

void LoadLetters () ;
/*--------------------------------------------------------------------------
        פונקציה הטוענת את הגופנים של האותיות למטריצה.
        פרמטרים מועברים - אין.
        ערך מוחזר - אין.
--------------------------------------------------------------------------*/

void PutDigit ( int dig ,int x , int y, byte color ,byte fx,byte *Page);
/* -------------------------------------------------------------------------
        פונקציה המציגה על המסך ספרה.
        פרמטרים מועברים :
            - dig   - איזו ספרה להציג .
            - x,y   - קואורדינטות של הנקודה הימנית עליונה.
            - color - הצבע.
            - fx    -  אפקט גוונים (כן או לא).
            - Page  - באיזה דף לצייר.
        ערך מוחזר :  אין.
--------------------------------------------------------------------------*/
```

```
void PutLetter ( char let ,int x , int y, byte color ,byte fx,byte *Page);
/* --------------------------------------------------------------------------
        פונקציה המציגה על המסך אות.
        פרמטרים מועברים :
           - let   - איזו ספרה להציג .
           - x,y   - קואורדינטות של הנקודה הימנית עליונה.
           - color - הצבע.
           - fx    -  אפקט גוונים (כן או לא).
           - Page  - באיזה דף לצייר.
        ערך מוחזר :  אין.
--------------------------------------------------------------------------*/

void PutText (char st[],int x,int y,byte color,byte fx,byte *Page) ;
/*---------------------------------------------------------------------------
        פונקציה המציגה על המסך מחרוזת (תו לא מזוהה לא יוצג).
        פרמטרים מועברים :
           - st    - המחרוזת .
           - x,y   - קואורדינטות של הנקודה הימנית עליונה.
           - color - הצבע.
           - fx    -  אפקט גוונים (כן או לא).
           - Page  - באיזה דף לצייר.
        ערך מוחזר :  אין.
--------------------------------------------------------------------------*/

void GetPal(byte ColorNo , byte *R , byte *G , byte *B) ;
/*---------------------------------------------------------------------------
        פונקציה הבודקת כיצד בנוי צבע מסוים.
        (החוזק של צבעי הבסיס : אדום, כחול וירוק)
        פרמטרים מועברים :
           - ColorNo - מספר צבע .
           - B,G,R -  ערכי צבעי הבסיס  (By Address).
        ערך מוחזר - אין.
 -------------------------------------------------------------------------*/

void SetPal(byte ColorNo , byte R , byte G , byte B) ;
/*---------------------------------------------------------------------------
        פונקציה הקובעת ערכים חדשים לצבעי הבסיס של תא מסוים בפלטת הצבעים.
        פרמטרים מועברים :
           - ColorNo - מספר צבע .
           - B,G,R -  ערכי צבעי הבסיס  (By Address).
        ערך מוחזר - אין.
--------------------------------------------------------------------------*/
```

```
void GrabPalette(void) ;
/*-----------------------------------------------------------------------------
                              פונקציה הקולטת את הלוח הצבעים למטריצת לוח הצבעים.
                                                         פרמטרים מועברים - אין.
                                                              ערך מוחזר - אין.
-----------------------------------------------------------------------------*/

void FadeOut(void);
/*-----------------------------------------------------------------------------
                                                 פונקציה המבצעת Fade Out למסך.
                                                         פרמטרים מועברים - אין.
                                                              ערך מוחזר - אין.
-----------------------------------------------------------------------------*/

void FadeIn (void);
/*-----------------------------------------------------------------------------
                                                 פונקציה המבצעת  Fade In למסך.
                                                         פרמטרים מועברים - אין.
                                                              ערך מוחזר - אין.
-----------------------------------------------------------------------------*/

void HideScreen(void);
/*-----------------------------------------------------------------------------
                                     פונקציה המסתירה את המסך ע"י איפוס לוח הצבעים.
                                                         פרמטרים מועברים - אין.
                                                              ערך מוחזר - אין.
-----------------------------------------------------------------------------*/

void LoadPalette(char *FileName) ;
/*-----------------------------------------------------------------------------
                                              פונקציה הטוענת פלטת צבעים מקובץ.
                                                    פרמטרים מועברים - שם הקובץ.
                                                              ערך מוחזר - אין.
-----------------------------------------------------------------------------*/

void LoadPage (byte *Page1 , byte *Page2);
/*-----------------------------------------------------------------------------
                                        לדף 2 פונקציה המעתיקה את תוכן דף 1.
                                                    פרמטרים מועברים - 2 דפים.
                                                              ערך מוחזר - אין.
-----------------------------------------------------------------------------*/
```

מימוש היחידה

```
/***************************************************************************
 *                          V G A 2 5 6 . C                               *
 *                       ~~~~~~~~~~~~~~~~~~                               *
 *             יחידה לטיפול במסך גרפי באיכות 256 צבעים                    *
 ***************************************************************************/

////////////////////////////////////////////////////////////////////////////
//                           ה צ ה ר ו ת                                  //
////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <dos.h>
#include <mem.h>
#include <math.h>

#define SCREEN   WIDTH  320
#define SCREEN   HEIGHT 200

#define sgn(x)  ((x<0)?-1:((x>0)?1:0)) // מאקרו שמחזיר את סימנו של מספר(0,-1,1)

typedef unsigned char  byte;
typedef unsigned short word;
typedef unsigned long  dword;

byte *VGA=(byte *) MK   FP (0xA000,0);
                  // מצביע על תחילת האזור בזיכרון השמור לנתוני המסך

byte *VPage1 = NULL ;
                  // מצביע על תחילת האזור בזיכרון המיועד לדף וירטואלי מס' 1

byte *VPage2 = NULL  ;
                  // מצביע על תחילת האזור בזיכרון המיועד לדף וירטואלי מס' 2

byte Pall[256][3] ;
                  // מטריצה שתכיל לוח צבעים של 256 צבעים

byte Digits [10][7] ;
                  // מטריצה שתכיל גופנים של ספרות
byte Letters[30][8] ;
                  // מטריצה שתכיל גופנים של אותיות
```

```
///////////////////////////////////////////////////////////////////////////
//                           פ ו נ ק צ י ו ת                              //
///////////////////////////////////////////////////////////////////////////

void Set256Mode (void)
/*-------------------------------------------------------------------------
                                    פונקציה להפעלת מוד 13H (256 צבעים).
                                    פרמטרים מועברים - אין.
                                    ערך מוחזר - אין.
-------------------------------------------------------------------------*/
{
   union REGS regs ;
   regs.x.ax=0x13 ;               // פרמטר H המציין מוד 13 VGA256
   int86(0x10,&regs,&regs) ;      // הפעלת פסיקת המסך
}

void SetTextMode (void)
/*-------------------------------------------------------------------------
                                    פונקציה להפעלת מוד טקסט רגיל.
                                    פרמטרים מועברים - אין.
                                    ערך מוחזר - אין.
-------------------------------------------------------------------------*/
{
   union REGS regs ;
   regs.x.ax=0x03 ;               // פרמטר H המציין מוד טקסט 3
   int86(0x10,&regs,&regs) ;      // הפעלת פסיקת המסך
}

void OpenVPage (void)
/*-------------------------------------------------------------------------
                                    פונקציה המקצה מקום לדפים הווירטואליים.
                                    פרמטרים מועברים - אין.
                                    ערך מוחזר - אין.
-------------------------------------------------------------------------*/
{
   VPage1 = (byte *) calloc(64000,1) ;
   VPage2 = (byte *) calloc(64000,1) ;
```

```
   if ( VPage1==NULL || VPage2==NULL  )
   {
      SetTextMode ;
      printf (" אין מספיק זיכרון ") ;
      exit (1) ;
   }
}

void CloseVPage (void)
/*---------------------------------------------------------------------------
        פונקציה המשחררת את הזיכרון שהוקצה לדפים הווירטואליים.
        פרמטרים מועברים - אין.
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   free (VPage1) ;
   free (VPage2) ;
}

void WaitRetrace ()
/*---------------------------------------------------------------------------
        פונקציה המחכה שתותח האלקטרונים יהיה בתחילת השהיה.
        פרמטרים מועברים - אין.
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   byte al ;
      לולאה הקוראת מפורט daH ערך ובודקת את הסיבית הרביעית (b3) //
      אם ערכה הוא אחד לוגי, התותח נמצא בהשהיה, לכן חכה //
      עד שההשהיה תיגמר (אנחנו רוצים למצוא את תחילת ההשהיה) //
   do
   {
      al = inp (0x03DA) ;
      al &= 0x08 ;
   }  while (al) ;
      לולאה הקוראת מפורט daH ערך ובודקת את הסיבית הרביעית (b3) //
      אם ערכה הוא אפס לוגי, התותח נמצא במהלך צביעה, לכן //
      חכה עד שהצביעה תיגמר, אז תתחיל ההשהיה .            //
```

```
   do
   {
      al = inp (0x03DA) ;
      al &= 0x08 ;
   }  while (!al) ;
}

void PutPixel (int x, int y, byte color , byte *Page)
/* ---------------------------------------------------------------------------
        פונקציה השותלת פיקסל בהיסט המחושב על פי x ו y מתחילת המצביע Page.
        פרמטרים מועברים
          - x - מיקום אופקי.
          - y - מיקום אנכי.
          - color - צבע.
          - Page - היכן לצייר (מסך או דף וירטואלי).
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   Page [(y<<8)+(y<<6)+x] = color ;
   /*      כיון שהזיכרון הוא לינארי ולא מטריצה, כדי להגיע לנקודה הרצויה
        יש לכפול את מספר השורה (y) ברוחב המסך (SCREEN_WIDTH) ולהוסיף x.
        אך במקום לעשות : 320*y+x אני משתמש בכפל ע"י הזזת סיביות שהוא
        מהיר יותר  ( 320*y = 2^6*y + 2^8*y ). */
}

void DrawLine (int x1, int y1, int x2, int y2, byte color, byte *Page)
/*---------------------------------------------------------------------------
        פונקציה לציור קו על דף/מסך.
        פרמטרים מועברים :
          - x1,y1 -  קואורדינאטות הנקודה השמאלית עליונה.
          - x2,y2 -  קואורדינאטות הנקודה הימנית תחתונה.
          - color - הצבע.
          - Page - היכן לצייר ( מסך או וירטואלי ).
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   int i,dx,dy,sdx,sdy,x,y,px,py;

   dx = x2-x1;            //  דלתא x - המרחק האופקי בין הנקודות
   dy = y2-y1;            //  דלתא y - המרחק ה אנכי בין הנקודות

   sdx= sgn(dx);          // הסימן של דלתא x (בעזרת המאקרו sgn)
   sdy= sgn(dy);          // הסימן של דלתא y (בעזרת המאקרו sgn)
```

```
   dx = abs(dx);          // ערך מוחלט לדלתא x
   dy = abs(dy);          // ערך מוחלט לדלתא y

   x  = dy>>1;            // x מקבל את מחצית דלתא y
   y  = dx>>1;            // y מקבל את מחצית דלתא x

   px=x1;  py=y1;                // p מקבל את מיקום הנקודה השמאלית עליונה

   PutPixel(px,py,color,Page); // שתילת הנקודה השמאלית עליונה

   if (dx>=dy)                   // אם הקו יותר אופקי מאנכי
   {
      for(i=0;i<dx;i++)
      {
         y+=dy;
         if (y>=dx)
         {
            y-=dx;
            py+=sdy;
         }
         px+=sdx;
         PutPixel(px,py,color,Page);
      }
   }
   else                  // אם הקו יותר אנכי מאופקי
   {
      for(i=0;i<dy;i++)
      {
         x+=dx;
         if (x>=dy)
         {
            x-=dy;
            px+=sdx;
         }
         py+=sdy;
         PutPixel(px,py,color,Page);
      }
   }
}
```

```
void DrawEmptyRect (int x1, int y1, int x2, int y2, byte color, byte *Page)
/*---------------------------------------------------------------------------
    פונקציה לציור מלבן חלול על דף/מסך.
    פרמטרים מועברים :
      - x1,y1 - קואורדינאטות הנקודה השמאלית עליונה.
      - x2,y2 - קואורדינטות הנקודה הימנית תחתונה.
      - color - הצבע.
      - Page - היכן לצייר ( מסך או וירטואלי ).
    ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   DrawLine(x1,y1,x1,y2,color,Page) ;
   DrawLine(x1,y1,x2,y1,color,Page) ;
   DrawLine(x2,y2,x1,y2,color,Page) ;
   DrawLine(x2,y2,x2,y1,color,Page) ;
}

void DrawFilledRect (int x1, int y1, int x2, int y2, byte color, byte *Page)
/*---------------------------------------------------------------------------
    פונקציה לציור מלבן מלא על דף/מסך.
    פרמטרים מועברים :
      - x1,y1 - קואורדינטות הנקודה השמאלית עליונה.
      - x2,y2 - קואורדינטות הנקודה הימנית תחתונה.
      - color - הצבע.
      - Page - היכן לצייר ( מסך או וירטואלי ).
    ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   for (;y1<=y2;y1++)
      DrawLine(x1,y1,x2,y1,color,Page) ;
}

void ScreenColor (byte color, byte *Page)
/*---------------------------------------------------------------------------
    פונקציה הצובעת את הדף/מסך.
    פרמטרים מועברים :
      - color - הצבע.
      - Page - היכן לצייר ( מסך או וירטואלי ).
    ערך מוחזר - אין.
---------------------------------------------------------------------------*/
```

```
{
   DrawFilledRect(0,0,319,199,color,Page) ;
}
int TestBit (byte X, int pos)
/* ---------------------------------------------------------------------------
   פונקציה הבודקת האם סיבית מסויימת בבית היא במצב לוגי "אמת".
   פרמטרים מועברים :
      X - הבית לבדיקה.
      pos - איזו סיבית לבדוק.
   ערך מוחזר :  מצב הסיבית
---------------------------------------------------------------------------*/
{
   X >>= pos ;              // סובב את הבית כמספר הפעמים הדרוש
   return ( X &= 0x1 ) ;    // החזר את מצב הסיבית הראשונה
}

void LoadDigits ()
/*---------------------------------------------------------------------------
   פונקציה הטוענת את הגופנים של הספרות למטריצה.
   פרמטרים מועברים - אין.
   ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   FILE *pf ;
   pf = fopen ("digits.mft","rb");    // פתח את קובץ גופני הספרות לקריאה
   fread(Digits,sizeof(byte),70,pf) ; // קרא הכל לתוך המטריצה
   close(pf) ;
}

void LoadLetters ()
/*---------------------------------------------------------------------------
   פונקציה הטוענת את הגופנים של האותיות למטריצה.
   פרמטרים מועברים - אין.
   ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   FILE *pf ;
   pf = fopen ("letters.mft","rb");   // פתח את קובץ גופני האותיות לקריאה

   fread(Letters,sizeof(byte),240,pf); // קרא הכל לתוך המטריצה
   close(pf) ;
}
```

```
void PutDigit ( int dig, int x, int y, byte color, byte fx,byte *Page)
/* ------------------------------------------------------------------------
    פונקציה המציגה על המסך ספרה.
    פרמטרים מועברים :
     - dig   - איזו ספרה להציג .
     - x,y   - קואורדינטות של הנקודה הימנית עליונה.
     - color - הצבע.
     - fx    -  אפקט גוונים (כן או לא).
     - Page  - באיזה דף לצייר.
    ערך מוחזר :  אין.
------------------------------------------------------------------------*/
{
   int i , j ;
   for (i=0;i<7;i++)            // לכל שורה בספרה
   {
      for(j=0;j<4;j++)          // לכל טור בספרה
      {
      if (TestBit (Digits[dig][i],j))
         PutPixel (x-j,y+i,color,Page) ;
      }
      if (fx) color++ ;         // אם האפקט מופעל , שנה גוון .
   }
}

void PutLetter ( char let ,int x , int y, byte color ,byte fx,byte *Page)
/* ------------------------------------------------------------------------
    פונקציה המציגה על המסך אות.
    פרמטרים מועברים :
     - let   - איזו ספרה להציג .
     - x,y   - קואורדינטות של הנקודה הימנית עליונה.
     - color - הצבע.
     - fx    -  אפקט גוונים (כן או לא).
     - Page  - באיזה דף לצייר.
    ערך מוחזר :  אין.
------------------------------------------------------------------------*/
{
   int i , j  , pos ;
                        // מציאת האות במטריצה
   if (let >= 'א' && let <= 'ת' )
      pos = let - 'א' + 1 ;
   else if (let == ',')
         pos = 28 ;
```

```
      else if (let == '.')
            pos = 29 ;
         else pos = 0  ;

   for (i=0;i<8;i++)              //לכל שורה באות.
   {
      for(j=0;j<8;j++)            //לכל טור באות.
      {
      if (TestBit (Letters[pos][i],j))
         PutPixel (x-j,y+i,color,Page) ;
      }
      if (fx) color++ ;           // אם האפקט מופעל, שנה גוון .
   }
}

void PutText (char st[],int x,int y,byte color,byte fx,byte *Page)
/*---------------------------------------------------------------------------
         פונקציה המציגה על המסך מחרוזת (תו לא מזוהה לא יוצג).
         פרמטרים מועברים :
            - st     - המחרוזת .
            - x,y    - קואורדינטות של הנקודה הימנית עליונה.
            - color  - הצבע.
            - fx     - אפקט גוונים (כן או לא).
            - Page   - באיזה דף לצייר.
         ערך מוחזר : אין.
---------------------------------------------------------------------------*/
{
   int p;
   for(p=strlen(st)-1;p>=0;p--)            // מהסוף להתחלה (עברית)
   {
      if (st[p] >= '0' && st[p] <= '9')    // אם זוהי ספרה
      {
         PutDigit (st[p]-'0',x,y+1,color,fx,Page );
         x -= 6 ;
      }                                    // אם זו אות
      else
      {
         PutLetter (st[p],x,y,color,fx,Page ) ;
         x -= 9 ;
      }
      if (fx) color ++ ;
   }
}
```

```
void GetPal(byte ColorNo , byte *R , byte *G , byte *B)
/*---------------------------------------------------------------------
            פונקציה הבודקת כיצד בנוי צבע מסוים.
            (החוזק של צבעי הבסיס : אדום, כחול וירוק)
            פרמטרים מועברים :
              - ColorNo - מספר צבע .
              - B,G,R -  ערכי צבעי הבסיס  (By Address).
            ערך מוחזר - אין.
---------------------------------------------------------------------*/
{
   outp (0x03C7,ColorNo);  //  שלח לפורט 3C7H את מספר הצבע המבוקש
   *R = inp (0x03C9);      //  קרא מפורט 3C9H את עוצמת הצבע האדום
   *G = inp (0x03C9);      //  קרא מפורט 3C9H את עוצמת הצבע הירוק
   *B = inp (0x03C9);      //  קרא מפורט 3C9H את עוצמת הצבע הכחול
}

void SetPal(byte ColorNo , byte R , byte G , byte B)
/*---------------------------------------------------------------------
            פונקציה הקובעת ערכים חדשים לצבעי הבסיס של תא מסוים בפלטת הצבעים.
            פרמטרים מועברים :
              - ColorNo - מספר צבע .
              - B,G,R -  ערכי צבעי הבסיס  (By Address).
            ערך מוחזר - אין.
---------------------------------------------------------------------*/
{
   outp (0x03C8,ColorNo); //  שלח לפורט 3C8H את מספר הצבע המבוקש
   outp (0x03C9,R);       //  שלח לפורט 3C9H את עוצמת הצבע האדום
   outp (0x03C9,G);       //  שלח לפורט 3C9H את עוצמת הצבע הירוק
   outp (0x03C9,B);       //  שלח לפורט 3C9H את עוצמת הצבע הכחול
}

void GrabPalette(void)
/*---------------------------------------------------------------------
            פונקציה הקולטת את לוח הצבעים לתוך מטריצת לוח הצבעים.
            פרמטרים מועברים - אין.
            ערך מוחזר - אין.
---------------------------------------------------------------------*/
{
   int i ;
   for(i=0;i<256;i++)
      GetPal(i,&Pall[i][0],&Pall[i][1],&Pall[i][2]);
}
```

```
void FadeOut(void)
/*---------------------------------------------------------------------------
    פונקציה המבצעת Fade Out למסך.
    פרמטרים מועברים - אין.
    ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   byte ColBase [3];                    // מערך של 3 בתים המייצגים צבע
   int i,j ;

   // לולאה חיצונית עד 64 כי החוזק המקסימלי של צבע בסיס הוא 64
   for(i=0; i<64; i++)
   {
      WaitRetrace() ;
      for(j=0 ; j<256; j++)             // לכל תא צבע בלוח הצבעים
      {
      // קרא ערכי צבעי בסיס של התא
      GetPal(j,&ColBase[0],&ColBase[1],&ColBase[2]) ;

      // לכל אחד מצבעי הבסיס, אם הוא גדול מאפס הורד ממנו אחד.
      if (ColBase[0] > 0) ColBase[0]--;
      if (ColBase[1] > 0) ColBase[1]--;
      if (ColBase[2] > 0) ColBase[2]--;

      // קבע לוח צבעים מתוקן לתא
      SetPal(j,ColBase[0],ColBase[1],ColBase[2]);
      }
   }
}

void FadeIn (void)
/*---------------------------------------------------------------------------
    פונקציה המבצעת Fade In למסך.
    פרמטרים מועברים - אין.
    ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   byte ColBase [3];                    // מערך של 3 בתים המייצגים צבע
   int i,j ;

   // לולאה חיצונית עד 64 כי החוזק המקסימלי של צבע בסיס הוא 64
```

```
   for(i=0; i<64; i++)
   {
      delay (40) ;
      WaitRetrace () ;
      for(j=0 ; j<256; j++)              // לכל תא צבע בלוח הצבעים
      {
                                         // קרא ערכי צבעי בסיס של התא
      GetPal(j,&ColBase[0],&ColBase[1],&ColBase[2]) ;

           // . לכל אחד מצבעי הבסיס, אם הוא קטן מהמקור הגדל אותו באחד
      if ((ColBase[0] < Pall[j][0]) && (ColBase[0] < 63)) ColBase[0]++;
      if ((ColBase[1] < Pall[j][1]) && (ColBase[1] < 63)) ColBase[1]++;
      if ((ColBase[2] < Pall[j][2]) && (ColBase[2] < 63)) ColBase[2]++;

                                         // קבע לוח צבעים מתוקן לתא
      SetPal(j,ColBase[0],ColBase[1],ColBase[2]);
      }
   }
}

void HideScreen(void)
/*--------------------------------------------------------------------------
        פונקציה המסתירה את המסך ע"י איפוס לוח הצבעים.
        פרמטרים מועברים - אין.
        ערך מוחזר - אין.
--------------------------------------------------------------------------*/
{
   int i ;

   for (i=0;i<256;i++)
      SetPal (i,0,0,0);
}

void LoadPalette(char *FileName)
/*--------------------------------------------------------------------------
        פונקציה הטוענת פלטת צבעים מקובץ.
        פרמטרים מועברים - שם הקובץ.
        ערך מוחזר - אין.
--------------------------------------------------------------------------*/
```

```
{
   FILE *fp ;

   fp=fopen(FileName,"rb") ;
   fread (Pall,sizeof(byte),256*3,fp) ;
   fclose (fp) ;
}

void LoadPage (byte *Page1 , byte *Page2)
/*-----------------------------------------------------------
                          פונקציה המעתיקה את תוכן דף 2 לדף 1.
                                     פרמטרים מועברים - 2 דפים.
                                               ערך מוחזר - אין.
--------------------------------------------------------------*/
{
   fmemcpy(Page1,Page2,64000) ;
}
```

תוכנית לדוגמה

בתחילה, תוכנית הדוגמה מציירת על המסך מלבנים בצבעים שונים זה על גבי זה. המלבן הראשון בגודל המסך וכל מלבן אחריו קטן בפיקסל אחד מכל כיוון. במרכז המסך כתובות המילים "הוד-עמי" עם אפקט אלכסון.

לאחר לחיצה על מקש כלשהו מתחיל השלב השני, בו לוח הצבעים משתנה לגוונים שונים של כחול. בכל פעם הגוונים מתחזקים באחת, ונוצרת תחושה של תזוזה במסך.

זכרו שהציור שצויר על המסך אינו משתנה, אלא לוח הצבעים בלבד!

התוכנית תסתיים בלחיצה נוספת על מקש כלשהו.

הקבצים שיש לצרף לפרויקט כדי להריץ את הדוגמה הם: Vga256.c ו-Vga_exam.c. למידע נוסף קראו נספח ו'.

```
/*** VGA_EXAM.C ***/

#include <stdio.h>
#include <conio.h>
#include <mem.h>

#include "VGA256.h"

/*** הכרזה על משתנים חיצוניים הנחוצים מהיחידה - VGA256 ***/
```

```
extern byte *VGA ;              // הכרזה על מצביע המסך
extern byte *VPage1 ;           // הכרזת המצביע לדף וירטואלי מס' 1
extern byte *VPage2 ;           // הכרזת המצביע לדף וירטואלי מס' 2
extern byte Pall[256][3] ;      // הכרזת מטריצת לוח הצבעים

void Init (void)
///// אתחול
{
   Set256Mode() ;               // עבור למצב 256 צבעים
   GrabPalette();               // תפוס את לוח הצבעים הנוכחי לתוך המטריצה
   HideScreen() ;               // הסתר (השחר) את המסך
   OpenVPage() ;                // פתח את הדפים הווירטואליים
   LoadDigits();                // טען את גופני הספרות
   LoadLetters();               // טען את גופני האותיות
}

void Draw (void)
///// ציור על גבי דף וירטואלי 1
{
   int x,y=0;
   for (x=0;x<160;x++)          // המציירת מלבנים מובנים על כל הדף לולאה
   {
      DrawFilledRect(x,y,319-x,199-y,x+110,VPage1);
      y++ ;
   }
   PutText("הוד עמי",190,95,36,1,VPage1);
                                // כתוב הוד עמי במרכז הדף עם אפקט
}

void Show (void)
///// הצגת הציור שצירנו בדף וירטואלי 1
{
   LoadPage(VGA,VPage1);        // טען את תוכן הדף למסך
   FadeIn () ;                  // חשוף את המסך באפקט Fade
}

void Close (void)
///// יציאה מסודרת מהתוכנית
{
   FadeOut () ;                 // עמעם את המסך באפקט Fade
   SetTextMode() ;              // חזור למצב טקסט
}
```

```
void PlayWithPalette (void)

///// יצירת אפקטים על ידי שינוי ערכי לוח הצבעים
{
   int i,j=0 ;
   FadeOut();
   while (!kbhit())       // כל עוד לא נלחץ שום מקש
   {
      WaitRetrace() ;     // המתן להשהיית תותח האלקטרונים
      j++ ;
      delay(50);          // השהיה

                          // שתי לולאות המאפסות את לוח הצבעים לגוונים של כחול
בלבד
                          // ערכי צבעי הבסיס אדום וירוק מאופסים וערך הכחול
משתנה בכל תא
      for (i=3;i<65;i++)
         SetPal(100+i+j,0,0,i);
      for (i=0;i<55;i++)
         SetPal(162+i+j,0,0,64-i);
   }
   getch () ;             // קלוט את המקש שנלחץ ביציאה מן הלולאה
}

void main (void)
{
   Init ();               // אתחול
   Draw ();               // ציור התמונה על גבי דף וירטואלי
   Show ();               // הצגת הדף הווירטואלי עם לוח הצבעים המקורי
   getch();               // המתן ללחיצת מקש כלשהו
   PlayWithPalette();     // שינוי ערכי לוח הצבעים בגוונים של כחול
   Close();               // יציאה
}
```

פרק 10

שילוב תמונות בפורמט BMP

בפרק זה נלמד לשלב תמונות מקבצים חיצוניים בפרויקט.

- ✔ **נכיר את פורמט "*BMP 256*" של Windows.**
- ✔ **נלמד את מבנה הכותר בקובץ BMP ואת תוכנו.**
- ✔ **נלמד להציג תמונה על המסך.**
- ✔ **לסיום, נלמד לטפל בתמונות עם קטעים שקופים.**

10.1 קבצי BMP

כיום קיימים סוגים רבים של קבצי תמונות, כגון: JPG, TIF, PCX, GIF ועוד.

בפרק זה נעסוק בקבצי BMP כי הם הנפוצים ביותר ופשוטים לקריאה (כפי שנלמד בהמשך).

מהו מקור השם BMP?

BMP הוא קיצור של **Bit Map** (מפת סיביות). השם "**מפת סיביות**" נקבע בתקופה שהמסכים יכלו להציג רק צבע אחד מלבד שחור (לרוב ירוק), כדי לייצג פיקסל כלשהו די היה בסיבית אחת. אם ערכה 'אמת' (כלומר '1'), יש צורך לשתול בה פיקסל, ואם ערכה 'שקר' (כלומר '0'), אין צורך לשתול בה סיבית (לתזכורת, ראו את הגופנים שיצרנו בפרק הקודם).

כיום קיימים פורמטים שונים של קבצי BMP. מכיון שאנו עובדים במוד 13H (עם 256 צבעים), נתרכז בפורמט 256 צבעים הסטנדרטי של מערכת ההפעלה Windows.

כיצד בנוי קובץ BMP?

קובץ BMP בנוי משלושה חלקים:

1. **כותר** (Header), בו נמצאים נתונים רבים אודות התמונה, כגון: הפורמט, הגובה, הרוחב וכו'.
2. **לוח צבעים** (Color Palette), שעל פיו נקבעים 256 הצבעים להצגת התמונה בצבעיה המדויקים.
3. נתוני התמונה עצמה, שבהם כל פיקסל מיוצג על ידי בית אחד, המסמל את מספר הצבע.

10.2 קריאת קובץ BMP

לפני שמציגים את התמונה על המסך, עלינו לקרוא את נתוניה החשובים מתוך הקובץ.
תחילה עלינו להגדיר טיפוס רשומה חדש שיכיל נתוני תמונה:

```
struct BmpRec
   {
      word width ;              // רוחב התמונה
      word height ;             // גובה התמונה
      byte *data ;              // נתוני התמונה
   } ;
```

כעת נפתח את הקובץ ונבדוק אם זהו אכן קובץ מסוג BMP בפורמט 256 צבעים. לשם כך נבדוק את ערכם של שני התווים הראשונים בקובץ. אם הקובץ הוא מסוג BMP, שני התווים הראשונים בכותר שלו יהיו BM (שימו לב שמדובר באותיות גדולות).

```
FILE *fp ;
word num_colors;
int x , j ;

///// פתיחת הקובץ
if ((fp = fopen(FileName,"rb")) == NULL)
   {
      printf("פתיחת קובץ BMP נכשלה");
      return (0) ;
   }

///// בדיקה האם זהו קובץ BMP תקין /////
if (fgetc(fp)!='B' || fgetc(fp)!='M')
   {
      fclose(fp);
      printf("הקובץ אינו קובץ BMP תקין");
      return(0);
   }
```

קריאת הנתונים מכותר הקובץ

בכותר הקובץ יש נתונים רבים שאינם נחוצים לנו, לכן נדלג בו אל הנתונים הדרושים (בעזרת הפקודה fseek), ונשמור את ערכיהם ברשומת תמונה. נעשה זאת כך:

```
fseek (fp,16*sizeof(byte),SEEK_CUR);      // קפוץ 16 בתים קדימה
fread(&bitmap->width,sizeof(word),1,fp); // קרא את רוחב התמונה
fseek (fp,2*sizeof(byte),SEEK_CUR);       // קפוץ 2 בתים קדימה
fread(&bitmap->height,sizeof(word),1,fp);// קרא את גובה התמונה
fseek (fp,22*sizeof(byte),SEEK_CUR);      // קפוץ 22 בתים קדימה
fread(&num_colors,sizeof(word),1,fp);     // קרא את מספר הצבעים
fseek (fp,6*sizeof(byte),SEEK_CUR);       // קפוץ 6 בתים קדימה
if (num_colors==0) num_colors=256;
```

כאן מסתיים כותר הקובץ, ומייד אחריו מתחיל לוח הצבעים.

קריאת לוח הצבעים מהקובץ

מבנה לוח הצבעים השמור בקובץ, שונה במקצת מלוח הצבעים שאנו עובדים איתו (כי פורמט BMP מיועד למערכת ההפעלה Windows). כל צבע מתוך 256 הצבעים בלוח הצבעים מיוצג על ידי 4 בתים. שלושת הבתים הראשונים מייצגים את עוצמת צבעי הבסיס (אדום, ירוק, כחול) כפי שהכרנו עד כה. עם זאת, **שימו לב!** עלינו לחלק את העוצמה ב-4 כדי להתאימה ללוח הצבעים שלנו (נעשה זאת בשיטת החלוקה המקוצרת על ידי הזזת סיביות כפי שלמדנו בפרק 2, "עבודה בסיביות").

```
for (j=0;j<256;j++)
   {
      Pall[j][2] = fgetc(fp) >>2 ;
      Pall[j][1] = fgetc(fp) >>2 ;
      Pall[j][0] = fgetc(fp) >>2 ;
      fseek (fp,sizeof(byte),SEEK_CUR) ;
   }
```

אם אינכם רוצים לעדכן את לוח הצבעים, דלגו עליו בדרך זו:

```
fseek(fp,sizeof(byte)*4*num_colors,SEEK_CUR) ;
```

קריאת התמונה מהקובץ

אחרי לוח הצבעים כתובים בקובץ ערכי התמונה. כל פיקסל בתמונה מיוצג על ידי בית אחד המכיל מספר המסמל את צבעו.

כדי לשמור את ערכי הפיקסלים, עלינו להקצות לתמונה מקום בזיכרון. גודל חוצץ הזיכרון יהיה כמספר הפיקסלים בתמונה (מכפלת הרוחב והגובה). נעשה זאת כך:

```
if((bitmap->data=(byte *)
     malloc( (word)(bitmap->width*bitmap->height) ))== NULL  )
                                  // אם ההקצאה נכשלה
   {
      fclose(fp);
      printf("אין מספיק זיכרון");
      return (0);
   }
```

לאחר הקצאת המקום הדרוש לתמונה, עלינו לקרוא אותה מהזיכרון. נעשה זאת כך:

```
i=(bitmap->height-1)*bitmap->width ;
for( ; i>=0 ; I -= bitmap->width )
   for( x=0 ; x<bitmap->width;x++)
       bitmap->data[(word)i+x]=(byte)fgetc(fp);
```

בכך סיימנו לקרוא את הנתונים הדרושים לנו מהקובץ, ואפשר לסגור אותו.

```
fclose(fp);
```

10.3 הצגת התמונה

ציור התמונה על המסך, או על דף וירטואלי

לאחר קריאת הנתונים מהקובץ, ניתן להציג את התמונה על המסך או על דף וירטואלי, החל מהנקודה (x,y) המייצגת את הקודקוד השמאלי העליון (כך נוהגים בדרך כלל). לשם כך נגדיר שני משתנים בגודל word (מילה): אחד להיסט התמונה ואחד להיסט המסך או הדף הווירטואלי. משתנה נוסף בגודל בית יכיל את מספר הצבע של הפיקסל הנוכחי.

```
word PageOffset = (y<<8)+(y<<6) ;    // היסט המסך/הדף הווירטואלי
word PicOffset = 0 ;                 // היסט התמונה
byte data ;
```

עכשיו נסרוק פיקסל אחר פיקסל ונעתיק אותם מהתמונה אל המסך או הדף הווירטואלי.



שימו לב! היזהרו לא לחרוג מגבולות המסך או הדף, כי הדבר ישבש נתונים בזיכרון.

כדי לוודא שאין חריגה, נבדוק זאת לפני שנשתול את הפיקסל. קטע התוכנית ייראה כך:

```
for( j=0 ; j<bitmap->height ; j++)
   {
      for( i=0 ; i<bitmap->width ; i++,PicOffset++ )
      {
         data = bitmap->data[PicOffset];
         if (x+i<320)                    //אם לא חרגת מגבולות החוצץ
            Page[PageOffset+x+i] = data;

      }
      if (PageOffset<63679)
         PageOffset += 320 ; // עבור לשורה הבאה
      else return(1);            //צא מהפונקציה אם חרגת מתחת לגבולות המסך
   }
```

בזאת סיימנו לצייר את התמונה על המסך.

טיפול בחלקי תמונה שקופים

לרוב אנו רוצים להציג תמונה על הרקע, הקיים כבר במסך. כך, התמונה לא תכסה מלבן שלם במסך, אלא רק את חלקי התמונה שרוצים להסתיר. לשם כך נגדיר שצבע מספר 'אפס', או צבע מספר '255' יהיו צבעים שקופים.

כיצד נשתול צבעים שקופים? התשובה פשוטה מאוד, לא נשתול! לשם כך נרחיב את התנאי לשתילת הפיקסל בפונקציה המציירת את התמונה על המסך, ונשתול פיקסל בתנאי שאינו שקוף:

```
if (data!=255 && x+i<320 && data)
     Page[PageOffset+x+i] = data;
```

10.4 הנחיות עבודה

לסיכום, מספר הערות חשובות לתשומת לב:

- מכיון שאין יותר צורך ברשומת התמונה, זכרו לשחרר את המקום שהקצתם בזיכרון עבור נתוני התמונה.

```
free (bitmap->data) ;
```

- זכרו שמוד 13H מאפשר הצגת 256 צבעים בו-זמנית בלבד, לכן נצלו היטב את לוח הצבעים. לעיתים תרצו שחלק מהצבעים בלוח הצבעים יתאימו לתמונה שאתם רוצים

לטעון וחלק יתאימו לשאר האלמנטים המצויירים על המסך. לשם כך תוכלו ליצור קובץ לוח צבעים ולכלול בו ערכים כאוות נפשכם, ולטעון אותו בתחילת התוכנית.

- אל תחששו לשנות צבעים מסוימים בתמונה, כדי שיתאימו ללוח הצבעים שלכם ולא ללוח הצבעים המקורי של התמונה. לשם כך כתבו תוכנית המחפשת פיקסלים מצבע A בקובץ BMP ומחליפה אותם בצבע B.
- אם תנסו לטעון תמונות שגודלן אינו מתחלק ב-4 תגלו שהתמונה מתעוותת, או שאינה מצוירת כלל. הסיבה לכך היא, שפורמט זה של BMP מעוות את הסדר בערכי התמונה אם מספר הפיקסלים בשורה אינו מתחלק ב-4. דרך פשוטה לפתור זאת היא לדאוג מראש שכל התמונות תהיינה בגודל המתחלק ב-4. אם הגודל אינו מתחלק ב-4, יש למלא את החלק המיותר בצבע שהגדרתם כשקוף, תוכלו לעשות זאת בתוכנת הצייר של Windows למשל.
  אם ברצונכם להתחכם, נסו לכתוב פונקציה הבודקת אם גודל התמונה אינו מתחלק ב-4, ושותלת אוטומטית פיקסלים שקופים בתמונה, כדי להתאימה למכפלה הקרובה של 4.
  לחילופין, ניתן לכתוב פונקציה הדואגת לתקן את העיוות, שנגרם כתוצאה מהגודל שאינו תקין.
- סוגים אחרים של קבצי תמונה יותר מסובכים להצגה, כי ערכי התמונה בהם מכווצים, וכדי להציג אותם צריך לשחזר תחילה את הערכים המקוריים.
  כדי לטפל בסוג ספציפי של קבצים, עליכם לחפש מידע על מבנה הכותר שלהם ועל אלגוריתם הכיווץ. לפי מידע זה, ניתן בקלות לשנות את הפונקציות שהודגמו כאן, כדי שיתאימו לקובץ והפורמט שמחפשים. מקור המידע הפשוט והנרחב ביותר הוא כמובן האינטרנט.

10.5 יחידה להצגת קבצי BMP

ממשק היחידה

```
/****************************************************************************
 *                          V i e w   B M P . H                             *
 *              ממשק ליחידה להצגת תמונות השמורות בפורמט BMP                 *
 * הערות : 1) היחידה תציג על המסך כל תמונת BMP השמורה כ-"BMP צבעים 256"     *
 *        בפורמט הסטנדרטי של Windows, בתנאי שמידותיה יתחלקו ב-4             *
 *        2) היחידה תומכת בצבע שקוף (פיקסל שהוגדר כשקוף לא יישתל על המסך) *
 ****************************************************************************/


/////////////////////////////////////////////////////////////////////////////
//                         ה צ ה ר ו ת                                     //
/////////////////////////////////////////////////////////////////////////////
```

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <dos.h>
#include <mem.h>

#include "VGA256.h"

struct BmpRec                         // רשומת תמונת BMP
   {
      word width ;                    // רוחב התמונה
      word height ;                   // גובה התמונה
      byte *data ;                    // נתוני התמונה
   } ;
//////////////////////////////////////////////////////////////////////////////
//                               פ ו נ ק צ י ו ת                              //
//////////////////////////////////////////////////////////////////////////////

int ReadBMP (struct BmpRec *bitmap , char *FileName) ;
/*---------------------------------------------------------------------------
                              פונקציה הקוראת קובץ BMP לזיכרון.
                              פרמטרים מועברים :
                                bitmap - רשומת BMP.
                                FileName - שם הקובץ.
                              ערך מוחזר :
                                אמת אם הפתיחה התבצעה בהצלחה.
---------------------------------------------------------------------------*/

int DrawBMP (struct BmpRec *bitmap, int x, int y, byte *Page) ;
/*---------------------------------------------------------------------------
                              פונקציה המציגה תמונה על המסך.
                              פרמטרים מועברים :
                                bitmap - רשומת BMP.
                                x , y  -  שם הקובץ.
                              ערך מוחזר :
                                אמת אם ההצגה התבצעה בהצלחה.
---------------------------------------------------------------------------*/
```

```
void CloseBMP (struct BmpRec *bitmap ) ;
/*---------------------------------------------------------------------------
        פונקציה המשחררת את הזיכרון שהוקצה לתמונה.
        פרמטרים מועברים -  אין.
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/
```

מימוש היחידה

```
/*************************************************************************
 *                          V i e w  B M P . C                           *
 *             יחידה להצגת תמונות השמורות בפורמט BMP                     *
 *  הערות : 1) היחידה תציג על המסך כל תמונת BMP השמורה כ-"BMP צבעים 256"  *
 *     בפורמט הסטנדרטי של Windows, בתנאי שמידותיה יתחלקו ב-4.          *
 *     2) היחידה תומכת בצבע שקוף(פיקסל שהוגדר כשקוף לא יישתל על המסך)   *
 *************************************************************************/

////////////////////////////////////////////////////////////////////////////
//                         ה צ ה ר ו ת                                   //
////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <dos.h>
#include <mem.h>
#include "VGA256.h"

/*** הכרזה על משתנים חיצוניים הנחוצים מהיחידה - VGA256 ***/

extern byte *VGA ;              // הכרזה על מצביע המסך
extern byte *VPage1 ;           // הכרזת המצביע לדף וירטואלי מס' 1
extern byte *VPage2 ;           // הכרזת המצביע לדף וירטואלי מס' 2
extern byte Pall[256][3] ;      // הכרזת מטריצת לוח הצבעים

struct BmpRec                   // רשומת תמונת BMP
   {
      word width ;              // רוחב התמונה
      word height ;             // גובה התמונה
      byte *data ;              // נתוני התמונה
   } ;
```

```
///////////////////////////////////////////////////////////////////////////
//                           פ ו נ ק צ י ו ת                              //
///////////////////////////////////////////////////////////////////////////

int ReadBMP (struct BmpRec *bitmap , char *FileName)
/*-------------------------------------------------------------------------
    פונקציה הקוראת קובץ BMP  לזיכרון.
    פרמטרים מועברים :
      bitmap - רשומת BMP.
      FileName - שם הקובץ.
    ערך מוחזר:
      אמת אם הפתיחה התבצעה בהצלחה.
-------------------------------------------------------------------------*/
{
   FILE *fp ;
   long i ;
   word num_colors;
   int x , j ;

   ///// פתיחת קובץ BMP
   if ((fp = fopen(FileName,"rb")) == NULL)
   {
      printf                 ("פתיחת קובץ ה-BMP נכשלה");
      return (0) ;
   }
   ///// בדיקה האם זהו קובץ BMP תקין
   if (fgetc(fp)!='B' || fgetc(fp)!='M')
   {
      fclose(fp);
      printf                ("הקובץ אינו קובץ BMP תקין");
      return(0);
   }

   ///// קראו את הנתונים החשובים מהקובץ
   fseek (fp,16*sizeof(byte),SEEK CUR) ;        // קפוץ 16 בתים קדימה
   fread(&bitmap->width, sizeof(word), 1, fp);  //קרא את רוחב התמונה
   fseek (fp,2*sizeof(byte),SEEK_CUR) ;         // קפוץ 2 בתים קדימה
   fread(&bitmap->height,sizeof(word), 1, fp);  //קרא את גובה התמונה
   fseek (fp,22*sizeof(byte),SEEK_CUR) ;        // קפוץ 22 בתים קדימה
   fread(&num   colors,sizeof(word), 1, fp);    // קרא את מספר הצבעים
   fseek (fp,6*sizeof(byte),SEEK_CUR) ;         // קפוץ 6 בתים קדימה
   if (num   colors==0) num_colors=256;         // בהנחה שזהו קובץ 8 bit
```

```
   ///// קריאת לוח הצבעים
   for (j=0;j<256;j++)
   {
   ///// :אם ברצונך שלוח הצבעים יתעדכן לפי הלוח השמור בתמונה כתוב כך
    /*
      Pall[j][2] = fgetc(fp) >>2 ;
      Pall[j][1] = fgetc(fp) >>2 ;
      Pall[j][0] = fgetc(fp) >>2 ;
      fseek (fp,sizeof(byte),SEEK_CUR) ;     */

      ///// : אם אינך מעוניין שלוח הצבעים יתעדכן דלג על הלוח בקובץ כך
      fseek (fp,sizeof(byte)*4,SEEK_CUR) ;
   }
   ///// הקצאת מקום בזיכרון לתמונה
   if ((bitmap->data=(byte *)
   malloc((word)(bitmap->width*bitmap->height))) == NULL) // אם ההקצאה נכשלה
   {
      fclose(fp);
      printf       ("אין מספיק זיכרון");
      return (0);
   }
   ///// קריאת התמונה לזיכרון
   for( i=(bitmap->height-1)*bitmap->width ; i>=0 ;
   i-=bitmap->width )
      for( x=0 ; x<bitmap->width;x++)
         bitmap->data[(word)i+x]=(byte)fgetc(fp);
   ///// סגירת הקובץ
   fclose(fp);
   return (1) ;
}

int DrawBMP (struct BmpRec *bitmap ,int x,int y , byte *Page)
/*-----------------------------------------------------------------------------
                                        .פונקציה המציגה תמונה על המסך
                                                      : פרמטרים מועברים
                                          .BMP רשומת - bitmap
                                          .שם הקובץ  -  x , y
                                                           : ערך מוחזר
                                      .אמת אם ההצגה התבצעה בהצלחה
-----------------------------------------------------------------------------*/
```

```
{
   int i,j ;
   word PageOffset = (y<<8)+(y<<6) ;         // היסט המסך/הדף הווירטואלי
   word PicOffset = 0 ;                      // היסט התמונה
   byte data ;
   ///// בדיקה האם יש מספיק מקום לצייר את התמונה
   if ( x > 320 || y > 200 )
   {
      printf            ("אין מקום לציור התמונה") ;
      return (0) ;
   }
   ///// ציור התמונה
   for( j=0 ; j<bitmap->height ; j++)
   {
      for( i=0 ; i<bitmap->width ; i++,PicOffset++ )
      {
         data = bitmap->data[PicOffset];
         if (data!=255 && x+i<320 && data)
            Page[PageOffset+x+i] = data; // במידה והצבע שקוף - לא יושתל פיקסל
      }
      if (PageOffset<63679)  PageOffset += 320 ;
      else return(1);
   }
   return (1) ;
}

void CloseBMP (struct BmpRec *bitmap )
/*-------------------------------------------------------------------------
                  פונקציה המשחררת את הזיכרון שהוקצה לתמונה.
                  פרמטרים מועברים - אין.
                  ערך מוחזר - אין.
--------------------------------------------------------------------------*/
{
   free (bitmap->data) ;
}
```

תוכנית לדוגמה

התוכנית לדוגמה צובעת את המסך בפסים כחולים וסגולים לסירוגין. על רקע זה היא טוענת תמונת BMP מקובץ Comp_Guy.BMP, מציגה אותה על המסך וגם מתייחסת לפיקסלים שקופים.



הקבצים שיש לצרף לפרויקט כדי להריץ את הפקודה הם : Vga256.c, Viewbmp.c ו-Bmp_exam.c. למידע נוסף קראו נספח ו'.

```
/*** BMP_EXAM.C ***/

#include <stdio.h>
#include <conio.h>
#include <mem.h>

#include "VGA256.h"
#include "ViewBMP.h"

/*** VGA256 – הכרזה על משתנים חיצוניים הנחוצים מהיחידה ***/

extern byte *VGA ;              // הכרזה על מצביע המסך
extern byte *VPage1 ;           // הכרזת המצביע לדף וירטואלי מס' 1
extern byte *VPage2 ;           // הכרזת המצביע לדף וירטואלי מס' 2
extern byte Pall[256][3] ;      // הכרזת מטריצת הפלטה
struct BmpRec bmp ;             // רשומת תמונה
```

```
void Init (void)
///// אתחול
{
   Set256Mode() ;                 // עבור למצב 256 צבעים
   GrabPalette();                 // תפוס את הפלטה הנוכחית לתוך המטריצה
   HideScreen() ;                 // הסתר (השחירו) את המסך
   OpenVPage() ;                  // פתח את הדפים הווירטואליים
   ReadBMP(&bmp,"comp_guy.bmp");// טעינת התמונה לרשומת התמונה
}

void Draw (void)
///// 1 ציור על גבי דף וירטואלי
{
   int i;
   for (i=0;i<200;i+=2)                    // לכל שני פסים במסך
   {
      DrawLine(0,i,319,i,201,VPage1);      // צבע את הראשון בכחול
      DrawLine(0,i+1,319,i+1,204,VPage1);  // צבע את השני בסגול
   }
   DrawBMP(&bmp,70,20,VPage1);             // ציור התמונה על רקע הפסים
}

void Show (void)
///// 1 הצגת הציור שציירנו בדף וירטואלי
{
    LoadPage(VGA,VPage1);          // טען את תוכן הדף למסך
   FadeIn () ;                    // Fade חשוף את המסך באפקט
}

void Close (void)
///// יציאה מסודרת מהתוכנית
{
   FadeOut () ;                   // Fade עמעם את המסך באפקט
   SetTextMode() ;                // חזור למצב טקסט
   CloseBMP (&bmp) ;              // סגירת רשומת התמונה
}

void main (void)
{
   Init ();                       // אתחול
   Draw();                        // ציור על גבי דף וירטואלי
   Show ();                       // הצגת הדף הווירטואלי
   getch();                       // המתן ללחיצת מקש כלשהו
   Close();                       // יציאה
}
```

פרק 11

זמן

בפרק זה נלמד לקבל מידע אודות הזמן והתאריך הנוכחי.

✓ **נלמד לקרוא את הזמן והתאריך הנוכחי משעון מערכת DOS.**

✓ **נלמד לשנות את הזמן והתאריך שבשעון מערכת DOS.**

✓ **נלמד לבנות קוצבי זמן (timers) שונים.**

✓ **לסיום, נפלוש לפסיקה 8H וכך נוכל לבנות קוצב זמן שיעבוד ברמת החומרה.**

גם הפעם ניצור בעצמנו יחידת תוכנית לטיפול בזמן, לא נשתמש בתוכנית הספריה 'time.h' המסופקת עם המהדר. נעשה זאת מכמה סיבות:

- ביחידת הספריה המקורית מתקבלת השנה כשדה (tm_year) בתוך רשומת זמן (tm), המחושב על פי הנוסחה: "1900 – (שנה נוכחית)", נוסחה זו גורמת לתאריכים שגויים, החל משנת 2000.
- בפרק זה נלמד ליצור קוצבי זמן שונים ברמת התכנות וברמת מערכת ההפעלה (באמצעות פסיקות), דברים שיחידת הספריה המקורית אינה תומכת בהם.
- מייד נגלה שאין זה מסובך לקבל מידע אודות הזמן והתאריך.

11.1 קריאת הזמן והתאריך

כמעט כל תוכנית מחשב משתמשת בנתוני הזמן והתאריך, לכן חשוב שנלמד כיצד ל'קרוא' נתונים אלה מהמערכת.

תחילה, עלינו להגדיר רשומה אחת שתכיל את נתוני הזמן, ורשומה שנייה שתכיל את נתוני התאריך:

```
struct TimeRec {            // רשומת זמן
  int H ;                   // שעות
  int M ;                   // דקות
  int S ;                   // שניות
  int HS ;                  // מאיות השנייה
     };
```

```
struct DateRec {            // רשומת תאריך
   int D ;                  // יום
   int M ;                  // חודש
   int Y ;                  // שנה
      };
```

קריאת הזמן הנוכחי משעון DOS

כדי לקרוא את הזמן הנוכחי משעון DOS יש להפעיל את פסיקת DOS, שמספרה 21H עם פרמטר 2c00H. לאחר הפעלת הפסיקה, ערכי האוגרים ישתנו באופן הבא:

אוגר ch יכיל את נתוני השעות.

אוגר cl יכיל את נתוני הדקות.

אוגר dh יכיל את נתוני השניות.

ואוגר dl יכיל את נתוני מאיות השנייה.

שימו לב! הערכים בשעון ובתאריכון DOS מיוצגים במבנה הקסדצימלי, אך אין צורך להמיר אותו, כי המהדר יעשה זאת באופן אוטומטי. זכרו, כאשר מדפיסים הודעות עם תו הבקרה d%, יודפס ערכו העשרוני של המשתנה, וכאשר משתמשים בתו הבקרה x%, יודפס הערך ההקסדצימלי (ראו פרק 2, "עבודה בסיביות").

אם כן, ניתן בקלות לכתוב פונקציה לקריאת הזמן הנוכחי ולהיעזר בפונקציה int86, כפי שלמדנו בפרק 3.

הפונקציה נראית כך:

```
void GetTime(struct TimeRec *Time)
{
 union REGS ireg ;

 ///// 2c00H הפעלת פסיקה 21 עם פרמטר
 ireg.x.ax =0x2c00 ;
 int86(0x21,&ireg,&ireg);

 ///// עדכון רשומת הזמן
 Time->H=ireg.h.ch ;
 Time->M=ireg.h.cl ;
 Time->S=ireg.h.dh ;
 Time->HS=ireg.h.dl ;
}
```

קריאת התאריך הנוכחי משעון DOS

כדי לקרוא את התאריך הנוכחי משעון DOS יש להפעיל את פסיקת DOS מספר 21H עם הפרמטר 2a00H. לאחר הפעלת הפסיקה, ערכי האוגרים ישתנו באופן הבא:

אוגר dl יכיל את היום הנוכחי.

אוגר dh יכיל את החודש הנוכחי.

אוגר cx יכיל את השנה הנוכחית.

הערה

שימו לב! נתוני השנה נכנסים לאוגר cx (אוגר כפול), כי לייצוג השנה דרושות 4 ספרות, ולא 2.

גם את הפונקציה לקריאת התאריך הנוכחי נממש בעזרת int86, כמו שעשינו במימוש הפונקציה הקודמת.



נסו לכתוב פונקציה המקבלת שתי רשומות זמן ומחזירה את ההפרש ביניהן בתוך רשומת זמן נוספת.
שימו לב ! התוצאה צריכה להיות חיובית. לדוגמה, אם ההפרש בין שדה השניות הוא שלילי, יש להפחית דקה משדה הדקות ולהוסיף 60 לשדה השניות. כמו כן, לגבי שדה הדקות והשעות.



נסו לכתוב פונקציה דומה המקבלת ומחזירה רשומות תאריך.
גם כאן יש לשים לב שההפרשים יהיו חיוביים.

11.2 עדכון הזמן והתאריך

עדכון הזמן הנוכחי בשעון DOS

לעיתים נרצה לשנות את השעה הנוכחית. גם לביצוע פעולה זו נשתמש בפסיקת DOS מספר 21H והפעם – עם הפרמטר 2d00h.

לפני הפעלת הפסיקה יש לעדכן את ערכי האוגרים באופן הבא:

אוגר ch צריך להכיל שעות.

אוגר cl צריך להכיל דקות.

אוגר dh צריך להכיל שניות.

אוגר dl צריך להכיל מאיות שנייה.

הפונקציה לעדכון השעה תיראה כך:

```
void SetTime(struct TimeRec Time)
{
  union REGS ireg ;

  ///// עדכון האוגרים בזמן הרצוי
  ireg.h.ch = Time.H;
  ireg.h.cl = Time.M;
  ireg.h.dh = Time.S;
  ireg.h.dl = Time.HS;

  ///// הפעלת פסיקה 21 עם פרמטר 2d00H
  ireg.x.ax =0x2d00 ;
  int86(0x21,&ireg,&ireg);
}
```

תזכורת: הפעם כתבנו ".Time" ולא "<-Time" כמו בפונקציה לקריאת השעה, משום שרשומת הזמן מועברת הפעם By value, ולא By Address באמצעות מצביע (ראו פרק 1, "תקציר פקודות בשפת C").

עדכון התאריך הנוכחי בשעון DOS

גם כשנרצה לעדכן את התאריך נשתמש בפסיקת DOS מספר 21H, והפעם – עם הפרמטר 2b00h. לפני הפעלת הפסיקה יש לעדכן את ערכי האוגרים באופן הבא:

אוגר dl צריך להכיל את היום.

אוגר dh צריך להכיל את החודש.

ואוגר cx צריך להכיל את השנה.

שימו לב! זכרו להכניס לאוגר cx את השנה בייצוג של 4 ספרות.

גם את הפונקציה לקריאת התאריך הנוכחי נממש בעזרת int86 כמו במימוש הפונקציה הקודמת.

11.3 יצירת קוצב שניות (טיימר שניות)

ברוב התוכניות והמשחקים יש שעון המתעדכן אחת לשנייה. כעת, כשאנו יודעים לקרוא את הזמן משעון DOS, נוכל בקלות לבנות קוצב זמן שניות, שיאפשר לעשות כל פעולה שנרצה בכל פעם שתחלוף שנייה אחת.

תחילה נכתוב פונקציה המחזירה את מצב השניות הנוכחי:

```
int Sec (void)
{
  union REGS ireg ;

  ///// 2c00H הפעלת פסיקה 21 עם פרמטר
  ireg.h.ah =0x2c ;
  int86(0x21,&ireg,&ireg);

  ///// החזר את מצב השניות
  return(ireg.h.dh);
}
```

ונגדיר משתנה גלובלי שיכיל את הערך בו השניות צריכות להיות בתקתוק הבא, כלומר ערך הגדול באחד מערך השניות הנוכחי:

```
int NextTick ;
```

משתנה זה צריך להתעדכן בכל שנייה, כדי שיכיל את הערך של השנייה הבאה. שימו לב, שאם השנייה הנוכחית היא 59, יש לעדכן את התקתוק הבא בערך 0 (ולא 60). נפתור בעיה זו באמצעות שיטת קידום "מודולו N" (הערך הבא הוא שארית החלוקה ב-60 של הערך הקודם + 1):

```
void UpdateNextTick (int S)
{
  NextTick = (S+1) % 60 ;
}
```

לאחר שמגדירים משתנה שיכיל את ערך השניות בתקתוק הבא, עלינו לכתוב פונקציה שתבדוק, האם ברגע זה מתרחש תקתוק. כלומר, האם ברגע זה ערך השניות שווה לערך שבמשתנה NextTick, ואם כן, נחזיר ערך 'אמת'.

```
int Tick (void)
{
  int S ;

  S = Sec();
```

```
  if (S==NextTick)                // אם יש תקתוק
    return(1);                    // החזר אמת

  else
    return(0);                    // אחרת החזר שקר
}
```

לסיום, נכתוב פונקציה המאתחלת את קוצב השניות (טיימר השניות) שבנינו, ומכינה אותו לפעולה על ידי שליפת ערך השניות הנוכחי, ועדכון המשתנה NextTick בערך הגדול ממנו ב-1.

```
void StartTimer (void)
{
  UpdateNextTick( Sec() );
}
```

יכולנו לבנות את קוצב זמן השניות בצורה הרבה יותר פשוטה, על ידי בדיקת ערך מאיות השנייה. בכל פעם שערך מאיות השנייה הוא אפס, פירושו שעברה שנייה.

אם כך, מדוע לא עשינו זאת?

אם היינו בוחרים בשיטה השנייה והתוכנית לא היתה בודקת את ערך מאיות השנייה בדיוק במאית השנייה בה מתחלפת השנייה, היינו "מאבדים" את התקתוק (ערך מאיות השנייה לא יהיה בדיוק אפס, ולכן התוכנית לא תזהה שהיה תקתוק). מכיוון שמטרתנו ליצור קוצב זמן שבין תקתוקיו, התוכנית תבצע פעולות אחרות, עלינו לוותר על השיטה השנייה הנוחה, כי אינה מתאימה ועלולה לגרום לאובדן מידע השינוי.

לעומת זאת, כשבודקים את ערך השניות, יש טווח של שנייה שלמה בה אפשר לבדוק את התקתוק, משום שלא מתייחסים לערך מאיות השנייה. משך זמן זה מספיק כמעט תמיד לזיהוי מעבר השנייה.

אם כך, בתוכנית הראשית נוכל לבנות לולאת המתנה כדוגמת הלולאה הבאה :

```
while (!Tick() && !Kbhit() && !MouseStatus(&x,&y));
```

לולאה זו מחכה לתקתוק, ללחיצת מקש או ללחיצה על לחצן העכבר, ורק כאשר אחד משלושת הגורמים הללו יחזירו 'אמת' - נצא מהלולאה ונוכל לבדוק מה גרם ליציאה מהלולאה (תקתוק, מקש או עכבר) ולפעול בהתאם.

ומה אם בכל זאת אנו צריכים לבצע פעולות שנמשכות יותר משנייה אחת? במקרה זה, קוצב זמן השניות ייעצר. צריך יהיה לאתחל אותו מחדש, לבדוק כמה זמן עבר ולעדכן את המונים המתאימים בתוכנית. לרוב, כשמשתמשים בקוצב זמן שניות, הפעולות המתבצעות במהלך התוכנית כתגובה ללחיצת מקש או לחצן בעכבר, נמשכות פחות משנייה (אם הפעולות נמשכות זמן רב יותר, סביר להניח שנכתבו בצורה לא יעילה).

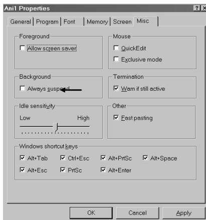
ומה נעשה אם איננו רוצים להציג מונה שניות על המסך, אלא רק למדוד את משך הביצוע של תהליך ממושך? במקרה זה נשמור ברשומת זמן את הזמן המדויק לפני ביצוע התהליך,

ברשומה נוספת נשמור את הזמן לאחר ביצוע התהליך, ונפעיל על שתי הרשומות האלו את פונקציית ההשוואה או הפרש הזמנים.

אך מה נעשה אם בכל זאת נצטרך להציג מונה שניות, או לבצע כל פעולה אחרת בזמן קבוע במהלך ביצוע של תהליך ממושך? במקרה זה, הטיפול בקוצב הזמן צריך להיות ברמת מערכת ההפעלה, ולא ברמת התכנות. בהמשך הפרק (סעיף 11.5) נלמד כיצד לעשות זאת.

הערה

לעיתים רוצים להפעיל את התוכנית תחת מערכת ההפעלה Windows בתוך חלון, ולא במסך שלם. כדי שקוצב זמן השניות יפעל כהלכה גם במצב זה, יש לבטל במאפייני קובץ ההרצה - exe (לחצן ימני על סמל הקובץ ובחירה במאפיינים). אם העכבר חורג מגבולות החלון אל הרקע, השהו תמיד את האפשרות. ראו תרשים 11.1.



תרשים 11.1

דוגמה לשימוש בקוצב זמן (טיימר) השניות, נמצאת בסוף הפרק.

11.4 השעון הפנימי (18.2Hz)

אם נרצה לבצע פעולה כלשהי בכל מספר קבוע של שניות ולא בכל שנייה, נוכל לעשות זאת בקלות בעזרת קוצב השניות שיצרנו. אך מה נעשה אם משך זמן ההמתנה אינו מתחלק במדויק בשניות?

אם במהלך ההשהיה, איננו רוצים שהתוכנית תבצע דברים אחרים, אנו יכולים להשתמש בפונקציות ההשהיה המוכרות של יחידת הספריה "dos.h":

delay(MS) - פונקציה המשהה את פעולת התוכנית למשך MS אלפיות השנייה (מילי שנייה).

sleep(S) - פונקציה המשהה את פעולת התוכנית למשך S שניות.

אך מה נעשה במקרה בו לא נרצה להשהות לחלוטין את ריצת התוכנית? לשם כך, נוכל להיעזר בקוצב הזמן הפנימי של המחשב הפועם בקצב של 18.2Hz, כלומר 18.2 פעמים בשנייה. ישנו מונה בזיכרון, שבכל פעימה כזו ערכו גדל באחד.

נגדיר מצביע על המונה הזה:

```
typedef unsigned long dword;
dword *Clock18 = (dword *) 0x0000046c ;
```

גודל המונה הוא dword, מילה כפולה חיובית, וידוע כי הערך המקסימלי של משתנה מסוג זה יכול להגיע ל- 4,294,967,295. אם נחלק ערך זה ב-18.2 (כי יש 18.2 פעימות בשנייה), נחלקו שוב ב-60 (60 שניות בדקה), נחלק שוב ב-60 (דקות בשעה) ונחלק שוב ב-24 (שעות ביממה), נגלה כי מונה זה יתאפס רק לאחר למעלה מ-2700 ימים! אם כך, מונה זה בהחלט אמין בשבילנו.

עכשיו נוכל לנצל את המצביע שיצרנו כדי לבצע פעולות בכל X פעימות (בקצב 18.2Hz). נעשה זאת בדרך דומה לזו שבה בנינו את קוצב זמן השניות. נקרא את ערך המונה הנוכחי ונוסיף לו ערך המייצג את כמות הפעימות, שאנו רוצים שיחלפו עד ש"השעון יצלצל" ונשווה בין שני הערכים עד לשוויון.

בפרק 13, "עכבר גרפי מונפש", נראה דוגמה לשימוש בקוצב 18.2Hz.

11.5 פלישה לפסיקה 8H

למדנו ליצור קוצב זמן של שניות וקוצב זמן שיפעל בעקבות פעימות מונה 18.2Hz. עם זאת, שני קוצבים אלה מהווים פתרון ברמת התכנות בלבד, ואם יש בתוכנית תהליכים ממושכים, קוצבי זמן אלה הם חסרי תועלת.

כדי להתגבר על הבעיה, עלינו למצוא פתרון ברמת מערכת ההפעלה!

שימו לב! מומלץ לקרוא תחילה את פרק 3.

בכל פעימה של קוצב הזמן הפנימי (18.2Hz), קורה דבר נוסף מלבד קידום המונה - מופעלת פסיקה 8H. אם נפלוש לפסיקה זו ונכתוב לתוכה תוכנית שירות (ISR) משלנו, נוכל לבצע כל פעולה שנרצה, לאחר כל פרק זמן מדויק המתחלק ב- 18.2Hz, בלי להתחשב במתרחש בתוכנית עצמה. ניתן לעשות זאת כי הפסיקה מופעלת ברמת מערכת ההפעלה ולכן היא יכולה לעצור את הרצת התוכנית הפעילה, כדי לבצע את הפקודות הכלולות בה (בתוכנית הפסיקה).

תחילה עלינו לשמור את הפסיקה המקורית בתא ריק, נניח 103, ולטעון לתוך תא 8 במערך הפסיקות את הפסיקה שלנו. נעשה זאת כך:

```
void SetIsr (void interrupt(*Isr)())  // הפונקציה מקבלת מצביע לפסיקה
{
   setvect(103,getvect(8));   // 103 שמור את הפסיקה המקורית בתא
   setvect(8,Isr);            // הכנס לתא 8 את הפסיקה שיצרת
}
```

נכין גם את הפונקציה שתשחזר את הפסיקה המקורית, ונפעיל אותה בסוף התוכנית.

```
void RestoreOldIsr (void)
{
   setvect(8,getvect(103));
}
```

עכשיו, כל שנותר לעשות הוא ליצור את תוכנית השירות לפסיקות (ISR) שלנו.

שימו לב! בתוך ה-ISR חייבת להיות קריאה לפסיקה המקורית (ששמרנו בתא 103), מכיון שייתכן שתוכניות אחרות הפועלות ברקע משתמשות בה.

אם איננו רוצים לבצע את הפעולה בתדירות של 18.2Hz, נגדיר מונה גלובלי שבו נכניס ערך, שיאמר בכל כמה פעימות אנו רוצים לבצע את פעולות תוכנית הפסיקה שלנו. בתוכנית השירות לפסיקות שניצור, נפחית את ערך המונה באחת ונבדוק אם ערכו אפס. אם כן - נבצע את הפעולות הרצויות ונאתחל אותו מחדש בערך הרצוי.

להלן תוכנית לדוגמה הפולשת לפסיקה 8H, ובכל 5 פעימות מציירת כוכבית על המסך. כלומר, כוכבית תצויר על המסך כ-3.64 פעמים בשנייה (18.2/5).

התוכנית נמצאת בתיקיה של פרק זה בדיסק המצורף (תחת התיקיה books\59281).

```
/**** HitIsr8.c ****/

#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define FREQ 5                 // 18.2Hz כל כמה "תקתוקים" של
int Mone ;
```

```
void interrupt MyIsr (void)      // הפסיקה החדשה שלנו
{
   union REGS ireg;
   int86(103,&ireg,&ireg);       // הפעלת הפסיקה המקורית ששמרנו בתא 103

   Mone--;
   if (!Mone)
   {
      putch('*');
      Mone = FREQ;
   }
}

void RestoreOldIsr (void)           // שחזור הפסיקה המקורית
{
   setvect(8,getvect(103));
}

void SetIsr (void interrupt (*Isr) () )
{
   setvect(103,getvect(8));        // שמור את הפסיקה המקורית בתא 103
   setvect(8,Isr);                 // הכנס לתא 8 את הפסיקה שיצרנו
}

void main (void)
{
   Mone = FREQ ;
   SetIsr(MyIsr);                  // טען את הפסיקה שיצרנו
   while(!kbhit());                // כל עוד לא נלחץ מקש כלשהו
   getch();
   RestoreOldIsr();                // שחזר את הפסיקה המקורית
}
```

11.6 שבב קוצב הזמן

למדנו ליצור קוצב זמן שאינו ברמת התכנות, אבל ראינו שהוא פועל בכפולות של 18.2Hz בלבד. אך מה נעשה אם נצטרך קוצב זמן שיפעל בתדירות יותר גבוהה?

כדאי לזכור שקצב של 18.2 פעמים בשנייה הוא מהיר למדי. אם אין לכם צורך בקוצב זמן בתדירות גבוהה מזו, אני מציע לדלג לסעיף הבא.

אם המשכתם לקרוא, אני מניח שהדיוק והמהירות באמת חשובים לכם. לשם כך, כדאי שתכירו את שבב קוצב הזמן שבמחשב (רכיב 8253-5).

בכל מחשב קיים שבב קוצב זמן הפועל בתדר של 1193180Hz, כלומר 1193180 פעימות בשנייה, וזה המון! בין תפקידיו הרבים של שבב זה, שאין זה המקום להרחיב עליהם, הוא להפעיל את הפסיקה 8 בתדר 18.2Hz. אבל מסתבר שתדר זה ניתן לתכנות על ידי פניה לפורט 40H ולפורט 43H.

כיצד מתכנתים את שבב קוצב הזמן, כדי שיפעיל את פסיקה 8 בתדר הרצוי? תחילה עלינו לחשב בכל כמה פעימות של שבב קוצב הזמן להפעיל את הפסיקה. נעשה זאת על פי הנוסחה הבאה: "תדר רצוי / תדר השבב = מספר פעמים". לדוגמה, אם רוצים שהפסיקה תופעל בתדירות של 200Hz, כלומר כל מחצית מאית שנייה, נציב כך את הנוסחה, ונקבל במשתנה mone את התוצאה הרצויה:

```
mone = 1193180 / 200 ;
```

כדי לתכנת את התדר החדש, נצטרך לבצע את הפעולות הבאות:

1. נשלח לפורט 43H את הערך 36H.
2. נשלח לפורט 40H את שמונה הסיביות הראשונות של המשתנה mone.
3. נשלח לפורט 40H את שמונה הסיביות האחרונות של המשתנה mone.

נעשה זאת כך (למידע נוסף, ראו פרק 2, "עבודה בסיביות"):

```
outportb (0x43,0x36);
outportb (0x40,mone&255);
outportb (0x40,mone>>8);
```

את שלוש הפקודות האלו נכלול בפונקציה הטוענת את ה-Isr שלנו (SetIsr).

כדי לאפס חזרה את השבב כך שיפעיל את פסיקה 8 בתדר ברירת המחדל 18.2Hz, נבצע את הפעולות האלו:

1. נשלח לפורט 443H את הערך 36H.
2. נשלח לפורט 40H את הערך 0.
3. נשלח לפורט 40H את הערך 0.

את שלוש הפקודות האלו נכניס לתוך הפונקציה המשחזרת את הפסיקה המקורית (RestoreOldIsr).

שימו לב! הפעם נוצר מצב הפוך. אנו רוצים שהפעולות יתבצעו בכל פעם שהפסיקה מופעלת ואילו את הפסיקה המקורית עלינו להפעיל רק בתדירות שלה. קודם ביצענו פעולה בכל כמה תקתוקים של 18.2 Hz, ואילו עכשיו אנחנו מבצעים פעולות בתדירות הגבוהה מ-18.2 Hz. לכן, כל מספר פעולות כאלו נפעיל את הפסיקה שתקדם את הקוצב המקורי הפועל בקצב של 18.2 Hz. אם כך, הפעם המונה יאותחל בערך "18 / תדר" ובכל פעם שערכו יהיה שווה לאפס, נפעיל את הפסיקה המקורית (ששמרנו בתא 103).

11.7 יחידה לטיפול בזמן

ממשק היחידה

```
/*****************************************************************************
 *                             M y T i m e . H                                *
 *                            ~~~~~~~~~~~~~~~~~~                              *
 *                      ממשק היחידה לטיפול בזמן ותאריך                         *
 *****************************************************************************/

//////////////////////////////////////////////////////////////////////////////
//                           ה  צ  ה  ר  ו  ת                                //
//////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <dos.h>

typedef unsigned long dword;

struct TimeRec {                      // רשומת זמן
   int H ;    // שעות
   int M ;    // דקות
   int S ;    // שניות
   int HS ;   // מאיות השנייה
          };

struct DateRec {                      // רשומת תאריך
   int D ;    // יום
   int M ;    // חודש
   int Y ;    // שנה
          };
```

```
///////////////////////////////////////////////////////////////////////////
//                             פ ו נ ק צ י ו ת                           //
///////////////////////////////////////////////////////////////////////////

void GetTime(struct TimeRec *Time);
/*-------------------------------------------------------------------------
                                          .פונקציה לשליפת הזמן משעון הדוס
                פרמטרים מועברים - *Time  (By Address)  -  רשומת זמן.
                                                        .ערך מוחזר - אין
-------------------------------------------------------------------------*/

void SetTime(struct TimeRec Time);
/*-------------------------------------------------------------------------
                                          .פונקציה לעדכון הזמן בשעון הדוס
                              פרמטרים מועברים - Time  -  רשומת זמן.
                                                        .ערך מוחזר - אין
-------------------------------------------------------------------------*/

void GetDate(struct DateRec *Date);
/*-------------------------------------------------------------------------
                                        .פונקציה לשליפת התאריך משעון הדוס
             פרמטרים מועברים - *Date  (By Address)  -  רשומת תאריך.
                                                        .ערך מוחזר - אין
-------------------------------------------------------------------------*/

void SetDate(struct DateRec Date);
/*-------------------------------------------------------------------------
                                        .פונקציה לעדכון התאריך בשעון הדוס
                           פרמטרים מועברים - Date  -  רשומת תאריך.
                                                        .ערך מוחזר - אין
-------------------------------------------------------------------------*/

int Sec (void);
/*-------------------------------------------------------------------------
                                            .פונקציה לבדיקת מצב השניות
                                                  .פרמטרים מועברים - אין
                                                .ערך מוחזר - מצב השניות
-------------------------------------------------------------------------*/
```

```
void UpdateNextTick (int S);
/*-------------------------------------------------------------------------
                                             פונקציה לעדכון התקתוק הבא.
                                         פרמטרים מועברים - מצב השניות.
                                                        ערך מוחזר - אין.
-------------------------------------------------------------------------*/

int Tick (void);
/*-------------------------------------------------------------------------
                                                  פונקציה לבדיקת תקתוק.
                                                   פרמטרים מועברים - אין.
                                            ערך מוחזר - אמת אם יש תקתוק.
-------------------------------------------------------------------------*/

void StartTimer (void);
/*-------------------------------------------------------------------------
                                            פונקציה להפעלת טיימר השניות.
                                                   פרמטרים מועברים - אין.
                                                        ערך מוחזר - אין.
-------------------------------------------------------------------------*/
```

מימוש היחידה

```
/**************************************************************************
 *                            M y T i m e . C                             *
 *                           ~~~~~~~~~~~~~~~~~~                           *
 *                     מימוש היחידה לטיפול בזמן ותאריך                     *
 *************************************************************************/

///////////////////////////////////////////////////////////////////////////
//                          ה צ ה ר ו ת                                  //
///////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <dos.h>

typedef unsigned long dword;

dword *Clock18 = (dword *) 0x0000046c ; // 18.2 Hz מצביע לקוצב הזמן הפנימי
```

```
struct TimeRec {                         // רשומת זמן
   int H ;    // שעות
   int M ;    // דקות
   int S ;    // שניות
   int HS ;   // מאיות השנייה
          };

struct DateRec {                         // רשומת תאריך
   int D ;    // יום
   int M ;    // חודש
   int Y ;    // שנה
          };

int NextTick ;                           // התקתוק הבא

//////////////////////////////////////////////////////////////////////////////
//                              פ ו נ ק צ י ו ת                               //
//////////////////////////////////////////////////////////////////////////////

void GetTime(struct TimeRec *Time)
/*------------------------------------------------------------------------
                                    פונקציה לשליפת הזמן משעון DOS
                 פרמטרים מועברים - *Time (By Address)  - רשומת זמן.
                                                  ערך מוחזר - אין.
------------------------------------------------------------------------*/
{
   union REGS ireg ;

   ///// הפעלת פסיקה 21 עם פרמטר 2c00H
   ireg.x.ax =0x2c00 ;
   int86(0x21,&ireg,&ireg);

   ///// עדכון רשומת הזמן
   Time->H=ireg.h.ch ;
   Time->M=ireg.h.cl ;
   Time->S=ireg.h.dh ;
   Time->HS=ireg.h.dl ;
}
```

```
void SetTime(struct TimeRec Time)
/*----------------------------------------------------------------------
                                     פונקציה לעדכון הזמן בשעון דוס.
                          פרמטרים מועברים - Time - רשומת זמן.
                                                   ערך מוחזר - אין.
----------------------------------------------------------------------*/
{
   union REGS ireg ;
   ///// עדכון האוגרים בזמן הרצוי
   ireg.h.ch = Time.H;
   ireg.h.cl = Time.M;
   ireg.h.dh = Time.S;
   ireg.h.dl = Time.HS;
   ///// 2d00H הפעלת פסיקה 21 עם פרמטר
   ireg.x.ax =0x2d00 ;
   int86(0x21,&ireg,&ireg);
}

void GetDate(struct DateRec *Date)
/*----------------------------------------------------------------------
                                     פונקציה לשליפת התאריך משעון דוס.
          פרמטרים מועברים - Date* (By Address) - רשומת תאריך.
                                                   ערך מוחזר - אין.
----------------------------------------------------------------------*/
{
   union REGS ireg ;
   ///// 2a00H הפעלת פסיקה 21 עם פרמטר
   ireg.x.ax =0x2a00 ;
   int86(0x21,&ireg,&ireg);
   ///// עדכון רשומת הזמן
   Date->D=ireg.h.dl ;
   Date->M=ireg.h.dh ;
   Date->Y=ireg.x.cx ;
}

void SetDate(struct DateRec Date)
/*----------------------------------------------------------------------
                                     פונקציה לעדכון התאריך בשעון דוס.
                          פרמטרים מועברים - Date - רשומת תאריך.
                                                   ערך מוחזר - אין.
----------------------------------------------------------------------*/
```

```
{
   union REGS ireg ;
   ///// עדכון האוגרים בתאריך הרצוי
   ireg.h.dl = Date.D ;
   ireg.h.dh = Date.M ;
   ireg.x.cx = Date.Y ;
   ///// 2b00H הפעלת פסיקה 21 עם פרמטר
   ireg.x.ax =0x2b00 ;
   int86(0x21,&ireg,&ireg);
}

int Sec (void)
/*-----------------------------------------------------------------------
                                        .פונקציה לבדיקת מצב השניות
                                              .פרמטרים מועברים - אין
                                            .ערך מוחזר - מצב השניות
-----------------------------------------------------------------------*/
{
   union REGS ireg ;
   ///// 2c00H הפעלת פסיקה 21 עם פרמטר
   ireg.h.ah =0x2c ;
   int86(0x21,&ireg,&ireg);
   ///// החזר את מצב השניות
   return(ireg.h.dh);
}

void UpdateNextTick (int S)
/*-----------------------------------------------------------------------
                                        .פונקציה לעדכון התקתוק הבא
                                    .פרמטרים מועברים - מצב השניות
                                                    .ערך מוחזר - אין
-----------------------------------------------------------------------*/
{
   NextTick = (S+1) % 60 ;
}

int Tick (void)
/*-----------------------------------------------------------------------
                                              .פונקציה לבדיקת תקתוק
                                              .פרמטרים מועברים - אין
                                       .ערך מוחזר - אמת אם יש תקתוק
-----------------------------------------------------------------------*/
```

```
{
   int S ;

   S = Sec();
   if (S==NextTick)                 // אם יש תקתוק
      return(1);                    // החזר אמת
   else
      return(0);                    // אחרת החזר שקר
}

void StartTimer (void)
/*-----------------------------------------------------------------------
                                            פונקציה להפעלת טיימר השניות.
                                                  פרמטרים מועברים - אין.
                                                        ערך מוחזר - אין.
-----------------------------------------------------------------------*/
{
   UpdateNextTick( Sec() );
}
```

תוכנית לדוגמה

התוכנית לדוגמה משתמשת בטיימר השניות, שבו כל שנייה מציגה על המסך את הזמן הנוכחי. התוכנית תסתיים בהקשה על מקש כלשהו.

תזכורת ! במחרוזת הפלט השתמשנו בתו הבקרה "%.2d". בצורה זו הערך מוצג תמיד בשני ספרות. לדוגמה, בשעה 12 ו-29 שניות, המחרוזת שתוצג על המסך תהיה "12:00:29".

הקבצים שיש לצרף לפרויקט כדי להריץ את הדוגמה הם: Mytime.c ו-Timechk.c. למידע נוסף קראו את נספח ו'.

```
/**** timechk.c ****/

#include <stdio.h>
#include "MyTime.h"

void main (void)
{
   struct TimeRec t ;

   clrscr();
   StartTimer();                        // אתחול הטיימר
```

```
   while (!kbhit())                    // כל עוד לא נלחץ מקש כלשהו
   {
      while (!Tick() && !kbhit());     // המתן לתקתוק או לחיצה על מקש
      if (Tick())
      {
         UpdateNextTick(Sec());        // עדכון התקתוק הבא
         GetTime(&t);                  // בדיקת  הזמן הנוכחי
         gotoxy(35,12);
         printf("%.2d:%.2d:%.2d",t.H,t.M,t.S);
      }
   }
   getch();
}
```

פרק 12

אנימציה

בפרק זה נלמד ליצור רשימת אנימציה ולהציגה על המסך.

- ✓ **נלמד ליצור רשימה מקושרת של מסגרות (Frames), אשר מכנים אותן לעיתים גם בשם שקופיות (Slides).**
- ✓ **נלמד לחלק תמונת אנימציה למסגרות (Frames) מתחלפות היוצרות אשליית תנועה, ולשמור אותן ברשימה המקושרת.**
- ✓ **לסיום, נלמד כיצד להציג קטע אנימציה מתוך רשימת אנימציה.**

12.1 רשימת אנימציה

אנימציה מורכבת מרצף של מסגרות (Frames), המתחלפות בקצב כלשהו.

כמעט בכל תוכנת מחשב נוכל למצוא קטעי אנימציה. קטעים אלה ניתן לייצג בדרכים רבות כגון קבצי סרטים, אני בחרתי להראות לכם דרך פשוטה יותר לעשות זאת.

את המסגרות (שהן למעשה תמונות בסגנון שקופיות) נצייר זו לצד זו בתוך תמונה אחת ונשמור אותה בפורמט BMP. לדוגמה, אם רוצים להראות שעון שמחוגיו מסתובבים, נצייר שעונים רבים זה לצד זה, ובכל אחד מהם נצייר מחוג בזווית שונה (ראה תרשים 12.1).

את תמונת האנימציה נטען לתוך רשימה מקושרת של מסגרות (נחלק את התמונה השלמה למסגרות, וכל אחת מהן נשמור בחוליה אחרת ברשימה), כדי שנוכל להציג את קטע האנימציה.



תרשים 12.1

אם היינו מציירים כל מסגרת בתמונת BMP נפרדת, במקום לצייר את כולן בתמונה אחת, היינו מבזבזים מקום יקר בזיכרון, כי בכל תמונה BMP יש כותרת בתחילת הקובץ (ראו פרק קודם).

כיצד נייצג את רשימת המסגרות במחשב?

רשימה מקושרת היא אוסף של חוליות, המקושרות זו לזו על ידי מצביעים. כל חוליה ברשימה תהיה במבנה הבא:

```
struct AniRec {
        struct BmpRec  Pic ;  // (frame)תמונות המסגרת הנוכחית
        struct AniRec *next ; // המסגרת הבאה
        struct AniRec *prev ; // המסגרת הקודמת
      };
```

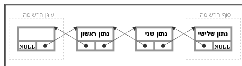
החוליה הראשונה ברשימה נקראת "עוגן" והחוליה האחרונה היא "סוף הרשימה". לפעמים חוליית העוגן מכילה נתונים, כמו שאר החוליות ברשימה. בדוגמה זו בחרתי שחוליית העוגן לא תכיל נתון כלשהו, אלא רק מצביע אל החוליה הבאה. נהגתי כך, מכיון שזו השיטה המקובלת (למשל, במקצוע "עיצוב תוכנה" הנלמד בבתי ספר שונים), אך בהחלט אפשר לכלול נתונים בחוליית העוגן.

הערה

אין זו בהכרח השיטה הפורמלית המוסכמת, ורבים דווקא אינם נוהגים כך. יתרון השיטה בכך שהיא חוסכת שימוש מסורבל ומבלבל של מצביע, המצביע על מצביע, המצביע על מצביע...ולכן היא יותר פשוטה ללימוד.

בדוגמה שלהלן כל חוליה מכילה מצביע אל החוליה הבאה וגם מצביע prev אל החוליה הקודמת. רשימה מסוג זה נקראת רשימה מקושרת דו-כיוונית (bidirectional list) והיתרון בה הוא, שאפשר לנוע ברשימה בשני הכיוונים, לפנים ולאחור.

בתרשים 12.2 מוצגת דוגמה לרשימה דו-כיוונית בעלת שלוש חוליות מידע.



תרשים 12.2

ברשימה שלנו, המידע הוא רשומת BMP בשם Pic.

12.2 יצירת חוליה לכל מסגרת

לאחר שהבנו כיצד בנויה תמונת האנימציה ורשימת האנימציה, עלינו לכתוב פונקציה המקבלת עוגן, תמונת אנימציה ומידע אודות רוחב כל מסגרת (Frame) בתמונת האנימציה. הפונקציה תחלק את תמונת האנימציה למסגרות ותקצה לכל אחת מהן חוליה אחת ברשימה.

```
int LoadAni (struct AniRec *ogen, char *FileName,
             word width)
{
  struct BmpRec bmp ;
  struct AniRec *node , *last, stam ;
  word i, NumOfFrames ;
```

Stam הוא משתנה מסוג חוליית אנימציה ששימושו היחידי הוא לסמל את גודל החוליה בהקצאת הזיכרון (כפי שתראו בהמשך).

ראשית, נגדיר כי סוף הרשימה מצביע על העוגן (אתחול הרשימה כריקה):

```
last = ogen;
```

לאחר מכן, עלינו לטעון את תמונת האנימציה לרשומת BMP:

```
if (!ReadBMP(&bmp,FileName))
  {
    printf (" תמונת האנימציה לא נמצאה ") ;
    return (0) ;                          // צא אם הטעינה נכשלה
  }
```

לאחר טעינת תמונת האנימציה, אנו יכולים לחשב כמה מסגרות יש בתמונה הזאת. נעשה זאת על ידי חלוקת רוחב התמונה (bmp.width) ברוחב המסגרת (width).

```
NumOfFrames = bmp.width /width ;
```

עכשיו, כשאנו יודעים כמה מסגרות יש לנו, נוכל לכתוב לולאת for שתרוץ על כל המסגרות.

```
for (i=1;i<=NumOfFrames;i++)
```

לכל מסגרת נבצע את הפעולות הבאות:

1. נקצה מקום בזיכרון לחוליה.

```
node = malloc (sizeof stam) ;
```

2. נעדכן את נתוני הרוחב והגובה של השדה Pic בחוליה.

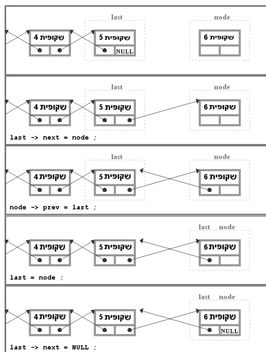
```
node->Pic.width = width ;
node->Pic.height = bmp.height ;
```

3. נקצה מקום בזיכרון לתמונת המסגרת בחוליה (pic.data).

```
node->Pic.data=(byte *) malloc((word) width *
                               bmp.height)
```

4. נכניס את החוליה שיצרנו לסוף הרשימה (ראה תרשים 12.3).

```
last -> next = node ;
node -> prev = last ;
last = node ;
last -> next = NULL ;
```


last
node
שקופית 4
שקופית 5
NULL
שקופית 6
last
node
שקופית 4
שקופית 5
שקופית 6
last -> next = node ;
last
node
שקופית 4
שקופית 5
שקופית 6
node -> prev = last ;
last node
שקופית 4
שקופית 5
שקופית 6
last = node ;
last node
שקופית 4
שקופית 5
שקופית 6
NULL
last -> next = NULL ;

תרשים 12.3

12.3 חלוקת תמונת האנימציה

לאחר שיוצרים חוליה לכל מסגרת ומחברים את החוליות לרשימה אחת, עלינו לחלק את התמונה עצמה למסגרות ולעדכן את נתוני התמונה בכל חוליה (Pic.data). נעשה זאת באמצעות הפקודה fmemcpy(dest,source,size) המעתיקה חוצץ זיכרון בגודל size מ-source אל dest. בכל פעם, נעתיק חוצץ בגודל שורה בשקופית, מתמונת האנימציה אל המסגרת שאליה השורה שייכת.

הגודל (size) יחושב כמכפלת אורך מסגרת בגודל בית ((width*sizeof(byte)).

המקור (source) ממנו נעתיק, יהיה החוצץ של תמונת האנימציה (bmp.data), בהיסט של מכפלת גודל שורה במספר ההעתקות שהתבצעו עד כה.

היעד (dest), יהיה חוצץ השקופית המתאימה, בהיסט של מכפלת גודל שורה במספר השורה (node->Pic.data+LineSize*I).

כך נראה קטע הקוד המבצע זאת:

```
LineSize = width * sizeof(byte) ;        // גודל שורה במסגרת
source = bmp.data ;   // תחילת המקור בתחילת נתוני תמונת אנימציה
for (i=0;i<bmp.height;i++)               // גובה המסגרת
{
   node = ogen->next ;                    // עבור אל המסגרת הראשונה
   for (j=0;j<NumOfFrames;j++)            // לכל מסגרת
   {
      dest = node->Pic.data+LineSize*i ; // עדכון היעד
      fmemcpy(dest,sorce,LineSize) ;      // העתקת שורה מהמקור ליעד

      node = node->next ;                 // עבור אל המסגרת הבאה
      source += LineSize ;               // קדם את המקור כגודל שורה
   }
}
```

12.4 מחיקת רשימת המסגרות

עד עתה למדנו לטעון תמונת אנימציה ולהפוך אותה לרשימה מקושרת של מסגרות. במהלך יצירת הרשימה הקצנו מקום רב בזיכרון, לכן בסוף השימוש ברשימה יש לשחרר את המקום שהוקצה עבורה.

כדי לשחרר את המקום שהוקצה לרשימה, צריך לשחרר חוליה אחר חוליה עד לסוף הרשימה. לפני שחרור כל חוליה, עלינו לשחרר גם את המקום שהוקצה לתמונה השמורה בה (על ידי הפעלת הפונקציה CloseBMP על השדה Pic בחוליה), רק אחר כך נשחרר את המקום המוקצה לחוליה עצמה.

שימו לב! כדי שנוכל לעבור לחוליה הבאה ברשימה, עלינו לשמור את כתובת החוליה הבאה, לפני שמשחררים את המקום שהוקצה לחוליה הנוכחית.

קטע הקוד נראה כך:

```
while (node)                         // עד סוף הרשימה
  {
    temp = node->next ;              // שמור את מיקום החוליה הבאה
    CloseBMP (&(node->Pic));         // סגור את תמונת המסגרת
    free (node) ;                    // שחרר את זיכרון החוליה
    node = temp ;                    // עבור אל החוליה הבאה
  }
```

12.5 סיכום

לסיכום, פירוט מלא של שתי הפונקציות, אחת לטעינת אנימציה ושנייה לסגירת אנימציה.

נושא זה מסתכם בשתי פונקציות בלבד, ולכן בחרתי לצרף אותן ליחידה "ViewBMP" ולא ליצור למענן יחידה נפרדת. היחידה המעודכנת נמצאת בתקליטור, בתיקיה של פרק זה (תחת התיקיה books\59281).

```
struct AniRec {                      // רשומת אנימציה
              struct BmpRec  Pic ;   // תמונת השקופית (Frame) הנוכחית
              struct AniRec *next ;  // המסגרת הבאה
              struct AniRec *prev ;  // המסגרת הקודמת
           }  stam;

int LoadAni (struct AniRec *ogen,char *FileName , word width)
/*-------------------------------------------------------------------------
                  פונקציה הטוענת אנימציה מתוך תמונה אל רשימת אנימציה.
                  פרמטרים מועברים :
                       ogen  -  עוגן רשימת האנימציה.
                   FileName  -  שם תמונת האנימציה.
                      width - רוחב מסגרת אחת בתמונת האנימציה.
                  ערך מוחזר :
                       אמת אם הטעינה עברה בהצלחה.
-------------------------------------------------------------------------*/
{
   struct BmpRec bmp ;
   struct AniRec *node , *last ,stam;
   word i, j , m, NumOfFrames , LineSize;
   byte *dest = NULL  ;                        // יעד
   byte *source = NULL  ;                      // מקור
```

```
if (! ReadBMP (&bmp,FileName))            // טען את תמונת האנימציה
   {
      printf ("תמונת האנימציה לא נמצאה") ;
      return (0) ;                        // צא אם הטעינה נכשלה
   }

last = ogen ;                             // סוף הרשימה מצביע על העוגן
NumOfFrames = bmp.width /width ;          // מספר השקופיות באנימציה

///// הכנת רשימת האנימציה, והקצאת חוליות לכל המסגרות
for (i=1;i<=NumOfFrames;i++)              // לכל מסגרת
{
   node = malloc (sizeof stam) ;          // הקצה זיכרון למסגרת
   if (node == 0)
   {
      printf ("אין מספיק זיכרון");
      return (0);                         // צא אם ההקצאה נכשלה
   }
   node->Pic.width = width ;              // עדכן את רוחב המסגרת
   node->Pic.height = bmp.height ;        // עדכן את גובה המסגרת

   if ((node->Pic.data=(byte *) malloc((word) width * bmp.height))
       == NULL)                           // אם ההקצאה נכשלה
   {
      printf ("אין מספיק זיכרון");
      return (0);
   }

   /////  הכנסת המסגרת לרשימת האנימציה
   last -> next = node ;
   node -> prev = last ;
   last = node ;
   last -> next = NULL ;
}
///// הכנסת נתוני התמונה לשקופיות הריקות שיצרנו ברשימה
LineSize = width * sizeof(byte) ;           // גודל שורה במסגרת
source = bmp.data ;     // תחילת המקור בתחילת נתוני תמונת האנימציה
for (i=0;i<bmp.height;i++)                  // כגובה מסגרת
{
   node = ogen->next ;                      // עבור אל המסגרת הראשונה
```

```
      for (j=0;j<NumOfFrames;j++)              // לכל מסגרת
      {
         dest = node->Pic.data+LineSize*i ; // עדכון היעד
            fmemcpy(dest,sorce,LineSize) ;  // העתקת שורה מהמקור ליעד
         node = node->next ;                 // עבור לשקופית הבאה
         source += LineSize ;        // קדם את מצביע המקור כגודל שורה
       }
   }
   CloseBMP (&bmp);
   return(1) ;
}

void CleanAniList (struct AniRec *ogen)
/*-------------------------------------------------------------------------
  פונקציה המשחררת זיכרון שהוקצה לרשימת אנימציה.
  פרמטרים מועברים :
  ogen - העוגן לרשימת האנימציה.
  ערך מוחזר - אין.
-------------------------------------------------------------------------*/
{
   struct AniRec *node , * temp ;
   node = ogen ;
   while (node)                              // עד סוף הרשימה
   {
      temp = node->next ;                    // שמור את מיקום החוליה הבאה
      CloseBMP (&(node->Pic));
      free (node) ;                          // שחרר את זיכרון החוליה
      node = temp ;                          // עבור לחוליה הבאה
   }
}
```

12.6 תוכנית לדוגמה

בעזרת התוכנית לדוגמה, תוכלו לראות כיצד משתמשים ברשימת האנימציה ומציגים את המסגרות על המסך. התוכנית טוענת תמונת אנימציה בעלת שתי מסגרות, שמצויר בהן יצור רץ (תרשים 12.4).



תרשים 12.4

כדי למנוע עיוותים, השתמשתי בשני דפים וירטואליים באופן הבא:

הרקע נשמר בדף מספר 2.

בכל צעד:

- הרקע מועתק מדף 2 לדף 1.
- המסגרת הנוכחית מצוירת על הרקע שבדף 1.
- דף 1 מועתק למסך.

הערה

לכל מי שלמד את המקצוע "עיצוב תוכנה": כאשר רצינו להגיע לנתון הראשון ברשימה כתבנו:

P ← עוקב ברשימה (עוגן רשימה (L), L)

קוראים זאת כך: המיקום ברשימה L של העוקב של עוגן הרשימה L מוכנס ל-P.

מכיון שהמסגרת הראשונה נמצאת בחוליה שאחרי העוגן, נממש זאת כך:

```
ManFrame = ManOgen->next ;
```

שימו לב! אל תתבלבלו בין משמעות החץ בביטוי האלגוריתם לבין משמעותו בביטוי המימוש. באלגוריתם, החץ מסמן פעולת הצבה לתוך מצביע P. לעומת זאת, בביטוי המימוש, החץ מסמל מצביע לשדה ברשומה.

הקבצים שיש לצרף לפרויקט כדי להריץ את הדוגמה הם: Vga256.c, Viewbmp.c ו-Ani1.c.
למידע נוסף קראו את נספח ו'.

```
/**** ani1.c ****/

#include <stdio.h>
#include <conio.h>
#include <mem.h>

#include "VGA256.h"
#include "ViewBMP.h"

/*** הכרזה על משתנים חיצוניים הנחוצים מהיחידה - VGA256 ***/

extern byte *VGA ;              // הכרזה על מצביע המסך
extern byte *VPage1 ;           // הכרזת המצביע לדף וירטואלי מס' 1
extern byte *VPage2 ;           // הכרזת המצביע לדף וירטואלי מס' 2
extern byte Pall[256][3] ;      // הכרזת מטריצת הפלטה
```

```
struct BmpRec bmp ;                     // רשומת תמונה
struct AniRec AniSize;
struct AniRec *ManOgen , *ManFrame;

void Init (void)
///// אתחול
{
   Set256Mode() ;                       // עבור למצב 256 צבעים
   LoadPallette("pall2.pal");           // טעינת לוח צבעים מקובץ
   HideScreen() ;                       // הסתר (השחר) את המסך
   OpenVPage() ;                        // פתח את הדפים הווירטואליים
   ReadBMP(&bmp,"bg1.bmp");             // טעינת תמונת הרקע לרשומת התמונה
   ManOgen= malloc (sizeof AniSize) ;   // הקצה מקום לעוגן
   LoadAni(ManOgen,"run1.bmp",100);     // טעינת תמונת האנימציה לרשימה
}

void Draw (void)
///// ציור על גבי דף וירטואלי 1
{
   ScreenColor(255,VPage1);
   DrawBMP(&bmp,0,0,VPage1);
   CloseBMP (&bmp) ;
}

void Close (void)
///// יציאה מסודרת מהתוכנית
{
   CleanAniList(ManOgen);
   FadeOut () ;                        // עמעם את המסך באפקט Fade
   SetTextMode() ;                     // חזור למצב טקסט
}

void main (void)
{
   int i=0;
   Init ();                            // אתחול
   Draw();                             // ציור על גבי דף וירטואלי
   FadeIn();
   ManFrame = ManOgen->next;           // הצבע על המסגרת הראשונה
   LoadPage(VPage2,VPage1);            // טעינת הרקע לדף 2
   while (i<320)                       // עד הקצה הימני של המסך
   {
      LoadPage(VPage1,VPage2);         // טען את הרקע מדף 2 ל-1
```

```
      DrawBMP(&(ManFrame->Pic),i,60,VPage1); // צייר את המסגרת
      LoadPage(VGA,VPage1);            // טען את דף 1 למסך
      if (ManFrame->next)              // אם לא סוף הרשימה
         ManFrame = ManFrame->next;
      else
         ManFrame = ManFrame->prev;

      i+=15;                           // זוז 15 פיקסלים ימינה
      delay(150);
   }
   Close();                            // יציאה
}
```



תרשים 12.5

בפרק הבא, תוכלו לראות דוגמה נוספת לרשימת אנימציה, המכילה יותר משתי מסגרות (או שקופיות, למי שנוהג לומר כך).

פרק 13

עכבר גרפי מונפש

בפרק זה נלמד להשתמש בעכבר במוד גרפי (graphic mode), וליצור לו מצביעים גרפיים מרשימים.

- ✔ **נלמד להתאים את פונקציות העכבר לרזולוציה של המוד הגרפי (13H).**
- ✔ **נלמד להחליף את המצביע המקורי בתמונת bmp.**
- ✔ **נלמד ליצור עכבר מונפש ("עכבר חושב").**

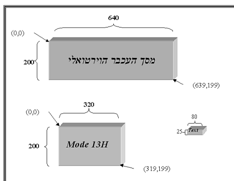
13.1 עכבר גרפי

בפרק 8, "עכבר למסך טקסט", למדנו להפעיל ולהשתמש בעכבר במוד טקסט. עשינו זאת בעזרת פסיקת העכבר 33H. כדי להתאים את הפונקציות שכתבנו בפרק 8 למוד גרפי 13H, עלינו להמיר את קואורדינטות העכבר לרזולוציה הדרושה במוד 13H.

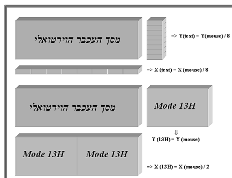
תזכורת! בפרק 8 גילינו שהמסך הווירטואלי של העכבר עובד ברזולוציה של 640x200. רזולוציית מסך הטקסט היא כזכור 80x25. לכן, כדי להמיר את קואורדינטות העכבר לקואורדינטות המסך הממשי, חילקנו את ערכי הקואורדינטות ב-8.

הפעם נתאים את הקואורדינטות לרזולוציה של מוד 13H שהיא 320x200. כיצד נעשה זאת? הרזולוציה האנכית היא 200, גם במסך העכבר הווירטואלי וגם במסך הממשי של מוד 13H, לכן אין צורך לבצע המרה. הרזולוציה האופקית של מוד 13H היא 320. שימו לב, ש-320 זה בדיוק מחצית מ-640 (הרזולוציה האופקית של העכבר), לכן נחלק את ערכי הקואורדינטות האופקיות ב-2.

תזכורת! ניתן לבצע חלוקה מהירה בחזקות של 2 בעזרת הזזת סיביות (כפי שלמדנו בפרק 2, "עבודה בסיביות").



תרשים 13.1



תרשים 13.2

13.2 יצירת מצביע לעכבר גרפי

אם ניקח את היחידה להפעלת עכבר טקסט שכתבנו בפרק 8 ונשנה את כל המרות הרזולוציה, כפי שלמדנו בסעיף הקודם, נוכל לראות שהפעם מצביע העכבר נראה כמו חץ לבן, ולא כמו מלבן כמו המצביע במוד טקסט. ניתן לשנות את צורת המצביע הגרפי בעזרת פסיקת העכבר 33H עם פרמטר 9, אך כל מה שניתן לשנות זה את צורת החץ. החץ יישאר לבן ופשוט.

כיצד נוכל ליצור מצביע מרשים יותר?

אם נטען תמונת BMP, שמצויר בה מצביע עכבר (כדוגמת), נוכל להשתמש בה כמצביע לעכבר במקום המצביע המקורי. נעשה זאת כך:

תחילה, נסתיר את המצביע המקורי של העכבר (HideMouse), אחר כך נמצא את הקואורדינטות הנוכחיות שלו, ובסוף נצייר את תמונת העכבר בקואורדינטות שמצאנו. כך למעשה אנו משתמשים במצביע בלתי נראה, ובמקום המצביע המקורי אנו רואים תמונת bmp.

כדי להציג את תמונת העכבר, נבצע את הפעולות הבאות:

1. נטען את הרקע מדף וירטואלי 1 לדף וירטואלי 2.

```
LoadPage(VPage2,VPage1);
```

2. נצייר את תמונת העכבר במיקום (MouseX,MouseY) על דף 2.

```
DrawBMP (bmp,MouseX,MouseY,VPage2);
```

3. נמתין להשהיית תותח האלקטרונים (כדי למנוע עיוותים).

```
WaitRetrace ();
```

4. לסיום, נטען את דף 2 אל המסך.

```
LoadPage(VGA,VPage2);
```

13.3 עכבר מונפש

בתוכנות ומשחקים רבים, לרבות מערכת ההפעלה Windows, כאשר המשתמש מפעיל תהליך ממושך, מצביע העכבר מתחלף במצביע מונפש (כדוגמת שעון החול של Windows). כך המשתמש יודע ש"המחשב עסוק" ואינו תקוע, או חסום.

כיצד נוכל להכין עכבר מונפש כזה?

בסעיף הקודם למדנו להחליף את המצביע המקורי של העכבר בתמונה כלשהי, ובפרק הקודם למדנו כיצד ליצור אנימציה. אם כך, כל שעלינו לעשות הוא להחליף בכל פעם את התמונה (שאנו מציגים במקום המצביע המקורי של העכבר), במסגרת (Frame) הבאה מרשימת האנימציה.

תחילה, עלינו להכין תמונת אנימציה ולצייר בה את המסגרות זו לצד זו. המצביע המונפש אינו חייב להיות שעון חול, וניתן להנפיש כל אובייקט שתרצו.

בתרשים 13.3 תוכלו לראות דוגמה לתמונת אנימציה של שעון אנלוגי.



תרשים 13.3

כדי להשתמש במצביע המונפש ובמצביע הדומם, עלינו להרחיב את פונקציית אתחול העכבר, כך שתבצע את הדברים הבאים: בדיקת תקינות העכבר, הסתרת המצביע המקורי של העכבר, טעינת תמונת העכבר הדומם לרשומת bmp בשם MouseCur, ולבסוף טעינת תמונת העכבר המונפש לרשימת אנימציה שבראשה העוגן MouseAniOgen.

הפונקציה תיראה כך:

```
int InstallMouse ( char *CurFileName , char *AniFileName )
{
 union REGS ireg ;
 int MouseAvailable ;                    //  האם קיים עכבר מחובר ומותקן?

 ///// בדיקת מספר הלחצנים בעכבר והאם הוא מחובר
 ireg.x.ax =0x00 ;
 int86 ( 0x33, &ireg , &ireg ) ;
 MouseAvailable = ireg.x.ax ;
 NumOfButtons = ireg.x.bx ;

if (!MouseAvailable)                   // אם אין עכבר מחובר ומותקן
   {
    printf (" לא נמצא עכבר מחובר ") ;
    return(0) ;
   }
if (!ReadBMP (&MouseCur,CurFileName)) // טען תמונת עכבר דומם
     {
    printf (" טעינת תמונת העכבר הדומם נכשלה ") ;
    return (0);                         // צא אם הטעינה נכשלה
   }

 ///// אתחול רשימת העכבר המונפש
 MouseAniOgen = malloc (sizeof tmp) ;
 if (MouseAniOgen == 0)
   {
    printf("אין מספיק זיכרון");
    return (0);
   }
```

```
 MouseAniOgen -> next = 0 ;
 MouseAniOgen -> prev = 0 ;
                                          // טעינת רשימת רשומות העכבר המונפש
 if  ( ! LoadAni(MouseAniOgen,AniFileName,12) )
   {
    printf ("טעינת תמונת העכבר המונפש נכשלה");
    return (0);                         // צא אם הטעינה נכשלה
   }
                          // שקופית האנימציה הראשונה היא בעוקב של העוגן
 MouseAniPic = MouseAniOgen -> next ;

 return (1);                            // אתחול העכבר התבצע בהצלחה
}
```

התוכנית הראשית צריכה להחליף את שקופיות העכבר המונפש בפרקי זמן קבועים. ניתן לעשות זאת בעזרת כל אחד מהטיימרים שהכרנו בפרק 11, "זמן". בתוכנית הדוגמה שבסוף הפרק, תוכלו לראות כיצד מטפלים בעכבר המונפש בעזרת מונה 18.2Hz.

13.4 היחידה לעכבר גרפי מונפש

קראו בתשומת לב את הממשק והמימוש, ושימו לב להבדלים בין תוכנית זו לבין התוכנית שבפרק 8. תוכנית הדוגמה שבסוף הפרק, תעזור לכם להבין כיצד הכל מתקשר וכיצד מתפעלים את העכבר הגרפי/מונפש.

ממשק היחידה

```
/***************************************************************************
 *                         G M o u s e . H                                 *
 *                        ~~~~~~~~~~~~~~~~~                                *
 *                  ממשק ליחידה לטיפול ושימוש בעכבר                        *
 ***************************************************************************/

/////////////////////////////////////////////////////////////////////////////
//                          ה צ ה ר ו ת                                    //
/////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <dos.h>
```

```
#define LBUTTON 0 ; // לחצן שמאלי
#define RBUTTON 1 ; // לחצן ימני
#define MBUTTON 2 ; // לחצן אמצעי

/*** הכרזה על משתנים חיצוניים הנחוצים מהתוכנית - VGA256 ***/

extern byte *VGA ;              // הכרזה על מצביע המסך
extern byte *VPage1 ;           // הכרזת המצביע לדף וירטואלי מס' 1
extern byte *VPage2 ;           // הכרזת המצביע לדף וירטואלי מס' 2

//////////////////////////////////////////////////////////////////////////////
//                          פ ו נ ק צ י ו ת                                 //
//////////////////////////////////////////////////////////////////////////////

int InstallMouse ( char *CurFileName , char *AniFileName ) ;
/*----------------------------------------------------------------------------
        פונקציה המתקינה ומאתחלת את העכבר
        פרמטרים מועברים :
          CurFileName - שם תמונת המצביע הדומם.
          AniFileName - שם תמונת המצביע המונפש.
        ערך מוחזר :
          אמת אם נמצא עכבר והוא אותחל בהצלחה.
----------------------------------------------------------------------------*/

int MouseRange ( int MinX, int MinY, int MaxX, int MaxY ) ;
/*----------------------------------------------------------------------------
        פונקציה להגדרת מרחב פעולת העכבר.
        פרמטרים מועברים :
         קואורדינטות נקודת הגבול המינימלי - MinX , MinY
         קואורדינטות נקודת הגבול המקסימלי - MaxX , MaxY
        ערך מוחזר : "אמת" אם התיחום התבצע בהצלחה.
----------------------------------------------------------------------------*/

void CancelRange ( void ) ;
/*----------------------------------------------------------------------------
         פונקציה לביטול התחום הנוכחי של העכבר.
         פרמטרים מועברים - אין.
         ערך מוחזר - אין.
----------------------------------------------------------------------------*/
```

```
void MouseStatus (int *button , int *x , int *y ) ;
/*--------------------------------------------------------------------------
פונקציה לבדיקת סטטוס העכבר ומיקומו.
פרמטרים מועברים :
x (By Address) - יקבל את מספר הטור עליו עומד המצביע.
y (By Address) - עומד המצביע יקבל את מספר השורה עליו.
button (By Address) - יקבל את מצב הלחצנים הנוכחי באופן הבא:
אף לחצן לא לחוץ - 0
לחצן שמאלי - 1
לחצן ימני - 2
לחצן שמאלי + ימני - 3
לחצן אמצעי - 4
לחצן שמאלי ואמצעי - 5
לחצן ימני ואמצעי - 6
שלושת הלחצנים לחוצים - 7
ערך מוחזר - אין
--------------------------------------------------------------------------*/

void MousePressPos ( int Button , int *X , int *Y ) ;
/*--------------------------------------------------------------------------
פונקציה למציאת מיקום הלחיצה האחרונה.
פרמטרים מועברים :
Button - איזה לחצן לבדוק (שמאלי-0, ימני-1, אמצעי-2)
X (By Address) - יקבל את מספר הטור
Y (By Address) - יקבל את מספר השורה.
ערך מוחזר - אין.
--------------------------------------------------------------------------*/

void MouseReleasePos ( int Button , int *X , int *Y ) ;
/*--------------------------------------------------------------------------
פונקציה למציאת מיקום השחרור האחרון.
פרמטרים מועברים :
Button - איזה לחצן לבדוק (שמאלי-0, ימני-1, אמצעי-2)
X (By Address) - יקבל את מספר הטור
Y (By Address) - יקבל את מספר השורה.
ערך מוחזר - אין.
--------------------------------------------------------------------------*/
```

```
void ShowMouse (int x , int y , struct BmpRec *bmp ) ;
/*---------------------------------------------------------------------------
        פונקציה המציגה מחדש את העכבר על המסך במיקום (x,y).
        פרמטרים מועברים :
          X  - המיקום האופקי של העכבר.
          Y  - המיקום האנכי של העכבר.
          bmp*  - מצביע לרשומת סמן העכבר הרצוי.
        ערך מוחזר : אין.
---------------------------------------------------------------------------*/

void CloseMouse (void) ;
/*---------------------------------------------------------------------------
        פונקציה הסוגרת את העכבר ( מפנה את הזיכרון שהוקצה ).
          ערך מועבר - אין.
          ערך מוחזר - אין.
---------------------------------------------------------------------------*/
```

מימוש היחידה

```
/***************************************************************************
 *                          M o u s e . C                                  *
 *                       ~~~~~~~~~~~~~~~~~~                                *
 *                    יחידה לטיפול ושימוש בעכבר                              *
 ***************************************************************************/

/////////////////////////////////////////////////////////////////////////////
//                       ה צ ה ר ו ת                                       //
/////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <dos.h>

#include "VGA256.h"
#include "ViewBMP.h"

#define LBUTTON 0 ; // לחצן שמאלי
#define RBUTTON 1 ; // לחצן ימני
#define MBUTTON 2 ; // לחצן אמצעי
```

```
/*** הכרזה על משתנים חיצוניים הנחוצים מהתוכנית - VGA256 ***/

extern byte *VGA ;                     // הכרזה על מצביע המסך
extern byte *VPage1 ;                  // הכרזת המצביע לדף וירטואלי מס' 1
extern byte *VPage2 ;                  // הכרזת המצביע לדף וירטואלי מס' 2

int NumOfButtons   ;                   // מספר הלחצנים בעכבר
struct BmpRec MouseCur ;               // רשומת תמונת המצביע הדומם
struct AniRec *MouseAniOgen,tmp ;      // עוגן לרשימת רשומות המצביע המונפש
struct AniRec *MouseAniPic ;           // מצביע לחוליה ברשימת האנימציה
int MouseX , MouseY ;                  // קואורדינטות העכבר
int MouseMode ;                        // מצביע דומם או מונפש ?
int Press ;                            // מצב לחצני העכבר

////////////////////////////////////////////////////////////////////////////
//                          פ ו נ ק צ י ו ת                                //
////////////////////////////////////////////////////////////////////////////

int InstallMouse ( char *CurFileName , char *AniFileName )
/*--------------------------------------------------------------------------
                                  פונקציה המתקינה ומאתחלת את העכבר
                                  פרמטרים מועברים :
                                  CurFileName  -  שם תמונת המצביע הדומם.
                                  AniFileName  -  שם תמונת המצביע המונפש.
                                  ערך מוחזר :
                                  אמת אם נמצא עכבר והוא אותחל בהצלחה.
--------------------------------------------------------------------------*/
{
   union REGS ireg ;
   int MouseAvailable ;                // האם קיים עכבר מחובר ומותקן?

   ///// בדיקת מספר הלחצנים בעכבר והאם הוא מחובר /////
   ireg.x.ax =0x00 ;
   int86 (0x33, &ireg , &ireg) ;       // קריאה לפסיקה 33 עם פרמטר 0
      /* כתוצאה מהפסיקה, אוגר AX יכיל "אמת" אם יש עכבר מחובר ומותקן
         במחשב, ואוגר BX יכיל את מספר הלחצנים של העכבר המחובר. */
   MouseAvailable = ireg.x.ax ;
   NumOfButtons = ireg.x.bx ;

   if (!MouseAvailable)                // אם אין עכבר מחובר ומותקן
      {
         printf ("לא נמצא עכבר מחובר");
         return(0) ;
      }
```

```
   if (!ReadBMP (&MouseCur,CurFileName)) // טען את תמונת העכבר הדומם
      {
         printf (" טעינת תמונת העכבר הדומם נכשלה ") ;
         return (0);                    // צא אם הטעינה נכשלה
      }
   ///// אתחול רשימת העכבר המונפש
   MouseAniOgen = malloc (sizeof tmp) ;
   if (MouseAniOgen == 0)
      {
         printf ("אין מספיק זיכרון");
         return (0);
      }
   MouseAniOgen -> next = 0 ;
   MouseAniOgen -> prev = 0 ;
                                 // טעינת רשימת רשומות העכבר המונפש
   if    ( ! LoadAni(MouseAniOgen,AniFileName,12) )
      {
         printf (" טעינת תמונת העכבר המונפש נכשלה ");
         return (0);                    // צא אם הטעינה נכשלה
      }
   MouseAniPic = MouseAniOgen -> next ;
   return (1);                          // אתחול העכבר התבצע בהצלחה
}

int MouseRange ( int MinX, int MinY, int MaxX, int MaxY )
/*-----------------------------------------------------------------------------
                                       פונקציה להגדרת מרחב פעולת העכבר.
                                                         פרמטרים מועברים :
                     MinX , MinY - קואורדינטות נקודת הגבול המינימלי
                     MaxX , MaxY - קואורדינטות נקודת הגבול המקסימלי
                                   ערך מוחזר : "אמת" אם התיחום התבצע בהצלחה.
-----------------------------------------------------------------------------*/
{
   union REGS ireg ;
   ///// בדיקת תקינות התחום
   if ( MinX<0 || MaxX>319 ||  MinY<0 || MaxY>199)
      return (0) ;                      // צא אם התחום חורג מגבולות המסך
   ///// התאמת הקואורדינטות למרחב העכבר
   MinX = MinX << 1 ;
   MaxX = MaxX << 1 ;
   ///// הגדרת התחום האופקי
   ireg.x.ax=0x7 ;
   ireg.x.cx = MinX  ;
```

```
   ireg.x.dx = MaxX  ;
   int86 ( 0x33 , &ireg , &ireg ) ;  קריאה לפסיקה 33 עם פרמטר 7 //
   הגדרת התחום האנכי /////
   ireg.x.ax=0x8  ;
   ireg.x.cx = MinY ;
   ireg.x.dx = MaxY ;
   int86 ( 0x33 , &ireg , &ireg ) ; קריאה לפסיקה 33 עם פרמטר 8 //
   הזזת הסמן לפינה השמאלית עליונה של התחום /////
   ireg.x.ax = 0x04 ;
   ireg.x.cx = MinX ;
   ireg.x.dx = MinY ;
   int86( 0x33 , &ireg , &ireg ) ;      קריאה לפסיקה 33 עם פרמטר 4 //
   return (1) ;
}

void CancelRange ( void )
/*---------------------------------------------------------------------------
                        פונקציה לביטול התחום הנוכחי של העכבר.
                        פרמטרים מועברים - אין.
                        ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
    MouseRange(0,0,319,199) ;
                  הפעלת פונקציית ההגבלה עם קואורדינטות גבולות המסך //
}

void MouseStatus (int *button , int *x , int *y )
/*---------------------------------------------------------------------------
                        פונקציה לבדיקת סטטוס העכבר ומיקומו.
                        פרמטרים מועברים :
        x (By Address) - יקבל את מספר הטור עליו עומד המצביע.
        y (By Address) - יקבל את מספר השורה עליו עומד המצביע.
        button (By Address) -  יקבל את מצב הלחצנים הנוכחי באופן הבא:
                        לחוץ אף לחצן לא - 0
                        לחצן שמאלי - 1
                        לחצן ימני - 2
                        לחצן שמאלי + ימני - 3
                        לחצן אמצעי - 4
                        לחצן שמאלי ואמצעי - 5
                        לחצן ימני ואמצעי - 6
                        שלושת הלחצנים לחוצים - 7
                        ערך מוחזר - אין
---------------------------------------------------------------------------*/
```

```
{
   union REGS ireg ;
   ireg.x.ax =0x03 ;
   int86 ( 0x33,&ireg,&ireg ) ; // קריאה לפסיקה 33 עם פרמטר 3
   *button = ireg.x.bx ;         // עדכון מצב הלחצנים
   *x = ireg.x.cx >>1 ;          // עדכון המיקום האופקי והמרה לערכי מסך גרפי
   *y = ireg.x.dx ;              // עדכון המיקום האנכי, אין צורך בהמרה
}

void MousePressPos ( int Button , int *X , int *Y )
/*---------------------------------------------------------------------------
            פונקציה למציאת מיקום הלחיצה האחרונה.
            פרמטרים מועברים :
            Button - איזה לחצן לבדוק (שמאלי-0, ימני-1, אמצעי-2)
            X (By Address) - יקבל את מספר הטור
            Y (By Address) - יקבל את מספר השורה.
            ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   union REGS ireg ;
   ireg.x.ax =0x05 ;
   ireg.x.bx = Button ;
   int86 ( 0x33, &ireg , &ireg ) ; // קריאה לפסיקה 33 עם פרמטר 5
   ///// עדכון הקואורדינטות
   *X = ireg.x.cx >> 1;
   *Y = ireg.x.dx ;
}

void MouseReleasePos ( int Button , int *X , int *Y )
/*---------------------------------------------------------------------------
            פונקציה למציאת מיקום השחרור האחרון.
            פרמטרים מועברים :
            Button - איזה לחצן לבדוק (שמאלי-0, ימני-1, אמצעי-2)
            X (By Address) - יקבל את מספר הטור
            Y (By Address) - יקבל את מספר השורה.
            ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   union REGS ireg ;
   ireg.x.ax =0x06 ;
   ireg.x.bx = Button ;
   int86 ( 0x33, &ireg , &ireg ) ; // קריאה לפסיקה 33 עם פרמטר 6
```

```
   ///// עדכון הקואורדינטות /////
   *X = ireg.x.cx >>1 ;
   *Y = ireg.x.dx ;
}

void ShowMouse (int x , int y , struct BmpRec *bmp )
/*-----------------------------------------------------------------------------
                         פונקציה המציגה מחדש את העכבר על המסך במיקום (x,y).
                                                           פרמטרים מועברים :
                                               X  - המיקום האופקי של העכבר.
                                               Y  - המיקום האנכי של העכבר.
                                      bmp*  - מצביע לרשומת סמן העכבר הרצוי.
                                                          ערך מוחזר : אין.
-----------------------------------------------------------------------------*/
{
   ///// עדכון הקואורדינטות החדשות של העכבר /////
   MouseX = x ;
   MouseY = y ;
   ///// ציור המצביע על המסך /////
   LoadPage(VPage2,VPage1);               // טען את הרקע
   DrawBMP (bmp,MouseX,MouseY,VPage2);    // צייר את העכבר
   WaitRetrace ();                        // המתינו להשהיית תותח האלקטרונים
   LoadPage(VGA,VPage2);                  // עדכן את המסך
}

void CloseMouse (void)
/*-----------------------------------------------------------------------------
                           פונקציה הסוגרת את העכבר(מפנה את הזיכרון שהוקצה)
                                                          ערך מועבר - אין.
                                                          ערך מוחזר - אין.
-----------------------------------------------------------------------------*/
{
   CleanAniList (MouseAniOgen) ;
}
```

תוכנית לדוגמה

תוכנית הדוגמה, מציגה על המסך את העכבר הגרפי בכל פעם שנלחץ לחצן העכבר. המצביע משתנה ממצביע מונפש (שזז בכל שתי פעימות של 18.2Hz) לבין מצביע דומם. התוכנית תסתיים עם הקשה על מקש Esc.

כדי לצייר את המצביע במקומו המתאים בכל פעם, התוכנית חייבת לעקוב אחרי כל שינוי בסטטוס העכבר. לשם כך משתמשים בשתי קבוצות של משתנים ששומרים את נתוני העכבר: משתנים גלובליים מתוך היחידה (MouseX, MouseY, Press), ומשתנים תואמים בתוכנית עצמה (x, y, p).

הלולאה העיקרית בודקת בכל פעם מהו סטטוס העכבר הנוכחי (הערכים מתעדכנים במשתנים p, x ו-y), ובודקת האם חל שינוי באחד מהפרמטרים האלה, על ידי השוואתם עם המשתנים הגלובליים של היחידה.

```
if ( Press!=p || MouseX!=x || MouseY!=y || *Clock18>=tic )
```

אם חל שינוי באחד הפרמטרים, התוכנית תגיב בהתאם ותעדכן את הפרמטרים הגלובליים.

שימו לב! נהוג שהתוכנה תגיב ללחיצת עכבר רק בזמן סיום הלחיצה (שחרור העכבר) ולא בזמן תחילת הלחיצה. לכן, עלינו לבדוק כי בפעם האחרונה סטטוס העכבר הראה כי הלחצן לחוץ, וכעת הלחצנים משוחררים. רק כאשר תנאי זה יתקיים, צריך להגיב ללחיצת העכבר (במקרה שלנו, נחליף בין מצביע דומם למצביע מונפש).

התנאי ייכתב כך:

```
if ( Press && !p )     // אם עכשיו הסתיימה לחיצה
```

יש לעקוב בתשומת לב אחר תוכנית הדוגמה, משום שהיא מסכמת למעשה את שש הפרקים האחרונים.

הקבצים שיש לצרף לפרויקט כדי להריץ את הדוגמה הם: Vga256.c, Viewbmp.c, Mytime.c, Gmouse.c ו-Anixmp.c. למידע נוסף קראו את נספח ו'.

```
/**** ANIxmp.C ****/

#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <dos.h>
#include <mem.h>
#include <time.h>

#include "VGA256.h"
#include "GMouse.h"
#include "ViewBMP.h"
```

```
#include "MyTime.h"

/*** VGA256 - הכרזה על משתנים חיצוניים הנחוצים מהתוכנית ***/

extern byte *VGA ;                    // הכרזה על מצביע המסך
extern byte *VPage1 ;                 // הכרזת המצביע לדף וירטואלי מס' 1
extern byte *VPage2 ;                 // הכרזת המצביע לדף וירטואלי מס' 2
extern byte Pall[256][3] ;            // הכרזת מטריצת הפלטה

/*** GMouse - הכרזה על משתנים חיצוניים הנחוצים מהתוכנית ***/

extern struct BmpRec MouseCur ;       // רשומת תמונת המצביע הדומם
extern struct AniRec *MouseAniOgen ;  // עוגן המצביע המונפש
extern struct AniRec *MouseAniPic ;   // שקופית נוכחית במצביע המונפש
extern int MouseX ,MouseY ;           // מיקום אחרון של המצביע
extern int Press ;                    // מצב הלחצנים האחרון
extern int MouseMode ;                // סוג המצביע (דומם או מונפש)

/*** MyTime - הכרזה על משתנים חיצוניים הנחוצים מהתוכנית ***/

extern dword *Clock18;                // מונה ה-18.2Hz

void Init (void)
{
   Set256Mode() ;
   HideScreen() ;
   LoadPallette ("Pall1.pal");
   OpenVPage() ;
   ScreenColor(145,VPage1) ;
   LoadDigits();
   LoadLetters();
   if (!InstallMouse("mouse.bmp","timer.bmp"))
      {
         printf("Mouse Not Found");
         exit(1);
      }
}

void main (void)
{
   int x,y,p,Esc=0;
   dword tic ;
```

```
   Init () ;                          // אתחול
   LoadPage (VGA,VPage1) ;            // טען את הרקע
   FadeIn () ;
   tic = *Clock18 +2 ;                // קבע את התקתוק הבא
   while (!Esc)
      {
        MouseStatus(&p,&x,&y) ;       // קרא את הנתונים הנוכחיים של העכבר
        if ( Press!=p || MouseX!=x || MouseY!=y || *Clock18>=tic )
                                      // אם השתנו נתוני העכבר או שיש "תקתוק"
        {
           if (*Clock18 >= tic)       // אם היה תקתוק
           {
              tic +=2 ;
              MouseAniPic = MouseAniPic->next ;
              if (!MouseAniPic)
                 MouseAniPic = MouseAniOgen->next ;
            }

            if ( Press && !p )        // אם עכשיו הסתיימה לחיצה
              MouseMode = !MouseMode ;

            Press=p;                  // עדכן מצב לחצנים

            if (MouseMode)            // אם המצביע מוגדר כמונפש
              ShowMouse(x,y,&(MouseAniPic->Pic)) ;
            else                      // אחרת
              ShowMouse(x,y,&MouseCur) ;

            if (kbhit())              // אם נלחץ מקש
            {
              if (getch() == 27)      // Esc אם נלחץ
                 Esc = 1;
            }
         }
      }

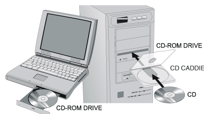
   ///// יציאה מסודרת מהתוכנית
   FadeOut() ;
   CloseVPage();
   CloseMouse() ;
   SetTextMode() ;
}
```

פרק 14

כונן התקליטורים (CD-ROM)

בפרק זה נלמד על כונן התקליטורים – CD-ROM.

- ✓ **נלמד לתקשר עם הכונן בעזרת MSCDEX.**
- ✓ **נלמד לקבל מידע אודות הכונן והתקליטור (הדיסק) שבתוכו.**
- ✓ **נלמד לשלוט על מגש הכונן.**
- ✓ **נלמד להשמיע שירים ולעבור לרצועה (שיר) מסוימת בתקליטור.**
- ✓ **נלמד לעקוב אחר ראש הכונן.**



14.1 מושגים כלליים

לפני שנוכל לתפעל את כונן התקליטורים, עלינו להכיר מספר מושגים הקשורים בו. אם כך, כדאי שנתחיל בשאלה הבסיסית ביותר:

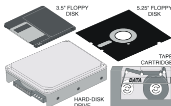
מה זה CD-ROM?

CD-ROM הם ראשי התיבות של "**C**ompact **D**isc **R**ead **O**nly **M**emory". מבנה כונן התקליטורים במחשב זהה למבנה הכוננים של מכשירי קומפקט דיסק המשמשים להשמעת תקליטורי מוסיקה. עם זאת, בתקליטורים המיועדים למחשב, הסיביות הצרובות יכולות להתפרש כנתונים ולא רק כמוסיקה.

יש מספר יתרונות לצורת אחסון הנתונים בתקליטור על פני צורות אחסון אחרות. למשל, אם נשווה את התקליטור לדיסקטים, נמצא שהוא יותר אמין מהם, כי לא יכול להיגרם לו נזק משדות מגנטיים, ומבחינת הנפח, הוא יכול להכיל כ-650MB של נתונים (כמות השווה לכ-450 דיסקטים). אם נשווה אותו לקלטת (קלטת מוסיקה, או קלטת של טייפ גיבוי במחשב) הוא הרבה יותר מהיר, אך אם נשווה אותו לדיסק קשיח, נמצא שנכון להיום הגישה לנתונים בדיסק הקשיח מהירה יותר.

מדוע למשל, כונן התקליטור אינו מהיר כמו כונן דיסק קשיח? הסיבה לכך היא שהתקליטורים יועדו במקור למוסיקה (המנוגנת בקצב קבוע), ולכן הנתונים צרובים בהם בצפיפות לינארית קבועה בכל מקום בדיסק. המשמעות היא, שבמסילות החיצוניות יש כמות גדולה יותר של נתונים מאשר במסילות הפנימיות. הגישה לנתוני המוסיקה צריכה להיות בקצב קבוע, ולכן מהירות הסיבוב (המהירות הקווית) של התקליטור היא קבועה. המהירות הזוויתית משתנה כשהראש האופטי זז, מכיון ששפת התקליטור כלפי פנים, או להפך. לכן, כשהדיסק מכיל נתונים המיועדים לגישה ישירה, יש צורך להאיץ ולהאט את המהירות לסירוגין כדי להגיע לנקודה הרצויה. לא ניכנס כאן לדיון פיסיקלי בנושאי מהירות מעגלית, אך נסכם שבסופו של דבר הצורך בהאצה והאטה גורמים לסיבוך, ולהאטת מהירות הגישה האפשרית לנתונים.

הדיסק הקשיח המגנטי מהיר יותר, כי הוא בנוי ופועל בצורה שונה. במסילות החיצוניות ובמסילות הפנימיות שלו יש כמות זהה של נתונים, לכן המהירות הזוויתית בו קבועה, ואין צורך בתאוצה והאטה הגורמים להאטת מהירות הגישה הישירה.



מה זה MSCDEX?

MSCDEX הם ראשי התיבות של "Microsoft CD-ROM Extensions" – הרחבות מיקרוסופט לכונן התקליטורים, ובפשטות: תוכנית שירות של מיקרוסופט להפעלת התקליטור. זהו למעשה דרייבר (driver), הנוסף למערכת ההפעלה (DOS או Windows), ומאפשר את הגישה לכונן התקליטורים. לאחר שקובץ זה נטען, ניתן לפנות לכונן התקליטור כאל כל כונן אחר במערכת.

אם כך, תוכנית הדרייבר MSCDEX דואגת לכך, שקבצי הנתונים שבתקליטור יהיו נגישים כמו בכל אמצעי אחסון אחר. אך מה קורה אם לפנינו תקליטור שמע (אודיו) המכיל מוסיקה ולא נתונים דיגיטליים? במקרה כזה, נצטרך להיעזר בפסיקה 2fH המתקשרת עם MSCDEX שמתקשרת עם כונן התקליטורים.

נשמע מסובך? אל דאגה, במהלך הפרק הדברים יתבהרו.

מיקרוסופט מספקת ללקוחותיה מידע בקובץ MSCDEX21.zip, המסביר את השימוש ב-MSCDEX. כל אדם שבונה כונני תקליטורים או כותב תוכנה המשתמשת בכונן כזה, צריך להתייחס למידע בקובץ זה. גם כל הכתוב בפרק שלפנינו (אמנם בתמצית) מסתמך על המידע מקובץ MSCDEX21. אם אתם זקוקים למידע נוסף, קראו בעצמכם את תוכן הקובץ, שניתן למצוא אותו באתרים רבים.

"ספרים צבעוניים" ותקני High Sierra

יש תקנים רבים העוסקים בגישה למיקום מסוים בתקליטור. בפרק זה נשתמש בשני תקנים: **Red Book** (**הספר האדום**) ו- **High Sierra**.

קבוצת תקנים אחת קרויה "הספרים הצבעוניים". בתקנים אלה, כל מקום בתקליטור מקבל ערך המייצג את הזמן שחלף בתנועת הראש הקורא. זהו הזמן הדרוש להשמעת התקליטור מתחילתו עד לנקודה המבוקשת.

אנו מכירים מספר סוגים שונים של ספרים צבעוניים:

- **Red Book** – הספר האדום: מיועד לתקליטורים המכילים רצועות (Tracks) מוסיקה.
- **Yellow Book** – הספר הצהוב: מיועד לתקליטורים המכילים קבצי נתונים.
- **Orange Book** – הספר הכתום: מיועד לתקליטורים הניתנים לצריבה.
- **Blue Book** – הספר הכחול: מיועד לתקליטורי לייזר.

אם הדבר נשמע לכם כמו בדיחה גרועה, או מזכיר לכם את המורה להיסטוריה מדבר על הספרים הלבנים, תראו מייד שזה ממש לא מסובך.

בפרק זה נלמד על תקליטורי מוסיקה, ולכן נשתמש בתקן של הספר האדום. בתקן זה כל מיקום בתקליטור מיוצג על ידי ערך בגודל מילה כפולה במבנה MMMMSSFF, המציין את זמן השמעה מתחילת התקליטור עד למיקום זה. כלומר, ארבע הספרות הראשונות מכילות את מספר הדקות שחולפות מתחילת התקליטור, שתי הספרות שאחריהן מכילות את מספר השניות שחולפות מסוף הדקה האחרונה, ושתי הספרות האחרונות מכילות את מספר המסגרות (frames), שחולפות מסוף השנייה האחרונה (יש 75 מסגרות בשנייה).

תקן נוסף החשוב לנו הוא High Sierra, או בשמו המקוצר **HSG**. על פי תקן זה, כל מקום על התקליטור מיוצג על ידי ערך **LSN** (Logical Sector Number), המייצג את מספר הסקטורים מתחילת התקליטור עד למיקום הנתונים. גם ערך זה מוצג בגודל מילה כפולה.

לשם מה עלינו להשתמש בשני התקנים, ואין אנו יכולים להסתפק בתקן אחד בלבד?

תקן HSG, מאפשר לבצע חישובי מיקום שונים. לדוגמה, כשאנו רוצים להשמיע שיר מסוים, עלינו "לומר" ל-MSCDEX כמה סקטורים ברצוננו להשמיע. כדי לדעת מהו אורך השיר, נשמור במשתנה אחד ערך LSN של מיקום תחילת השיר, ובמשתנה אחר – ערך LSN של מיקום סוף השיר. הפרש שני המשתנים הללו יהיה מספר הסקטורים שבהם מאוחסן השיר.

לעומת זאת, אם נרצה למסור למשתמש מידע אודות השיר, לא נכתוב לו את אורך השיר במבנה LSN, כי ערך זה חסר משמעות עבורו. לשם כך, נשתמש בתקן הספר האדום השומר את המיקום במבנה זמן. תחילה עלינו לפרק את מבנה הזמן (MMMMSSFF) לדקות, שניות ומסגרות. נעשה זאת על ידי הפונקציה הזו:

```
void TranslateRedBook(dword Value,word *M, byte *S,byte *F)
{
  *F = Value & 0x000000ff;              // פירוק ערך המסגרות
  *S = (Value & 0x0000ff00) >> 8;       // פירוק ערך השניות
  *M = (Value & 0xffff0000) >> 16;      // פירוק ערך הדקות
}
```

שימו לב! גודל משתנה הדקות הוא מילה, וגודל משתני השניות והמסגרות הוא בית. נכון להיום, כונני התקליטור יכולים להכיל מוסיקה בהיקף של כ-74 דקות, אך דבר זה עשוי להשתנות בעתיד. על כן, בתקן הספר האדום הוגדרו ארבע ספרות המייצגות את ערך הדקות ולא שתיים, ורצוי להשתמש במשתנה בגודל מילה ולא בגודל בית.

לאחר שלמדנו על ייצוג ערכים בשני התקנים, נכתוב פונקציות הממירות ערכים מתקן אחד לאחר. הפונקציה הממירה ערך בתקן הספר האדום לערך LSN, תיראה כך:

```
dword ConvertRedBookToLSN (dword Value)
{
  word Min;
  byte Sec, Frame;
  dword LSN;

  TranslateRedBook (Value, &Min, &Sec, &Frame);

  LSN = Frame;                    // מספר המסגרות
  LSN +=(dword) Sec * 75;         // + שניות (בשנייה 75 מסגרות)
  LSN += (dword) Min * 75 * 60;   // + דקות (60 שניות בדקה)
  LSN -= 150;

  return (LSN);
}
```

שימו לב! על פי המידע ב-MSCDEX21, כשממירים ערכי משתנים בין שני תקנים אלה, יש להפחית שתי שניות. על כן הפחתנו 150 מהערך LSN (150=2*75).

הפונקציה הממירה ערך LSN לערך על פי הספר האדום, תיראה כך:

```
dword ConvertLSNToRedBook (dword Value)
{
  word Min;
  byte Sec, Frame;
  dword Red;

  Value += 150;
  Frame =  Value % 75;       // מספר המסגרות העודפות לאחר חלוקה לשניות
  Value -= Frame;            // החסר את המסגרות העודפות
  Value /= 75;               // חלוקה לשניות (בשנייה 75 מסגרות)
  Sec   =  Value % 60;       // מספר השניות העודפות לאחר חלוקה לדקות
  Value -= Sec;              // החסר את השניות העודפות
  Min   =  Value / 60;       // חלוקה לדקות
  ///// MMMMSSFF הרכבת מבנה הזמן
  Red =  (dword) Frame;
  Red += ((dword) Sec) << 8;
  Red += ((dword) Min) << 16;

  return (Red);
}
```

בזאת סיימנו את ההקדמות המייגעות ואפשר להתחיל בתכנות עצמו.

14.2 בדיקת אתחול הדרייבר MSCDEX והפעלתו

בדיקת אתחול ה-MSCDEX

לפני שמתחילים לתקשר עם MSCDEX, עלינו לבדוק האם תוכנית הדרייבר טעונה בזיכרון, והאם יש בכלל כונן תקליטור במחשב.

לשם כך, נשתמש בפסיקה 2fH עם פרמטר 1500H. הפסיקה תחזיר אמת באוגר bx, אם הדרייבר MSCDEX מותקן ובמחשב קיים כונן תקליטורים. באוגר cx תוחזר אות הכונן.

נעשה זאת בעזרת הפונקציה int86, כפי שלמדנו בפרק 3. הפונקציה תיראה כך:

```
byte CheckMSCDEX (void)
{
   union REGS ireg ;

   ireg.x.ax = 0x1500;
   int86 (0x2F,&ireg,&ireg);

   if (ireg.x.bx)              // אם הדרייבר MSCDEX מותקן ונמצאו כונן
   {
      DiscInfo.Drive =ireg.x.cx ;        // שמור את אות הכונן
      return (1);                        // והחזר אמת
   }
   else return (0);                      // אחרת החזר שקר
}
```

שימו לב, שאת אות הכונן שמרנו ב-DiscInfo.Drive. DiscInfo - זו רשומה גלובלית, אשר נשמור בה נתונים שונים אודות הכונן והתקליטור שנמצא בו. הגדרנו אותה כגלובלית, מכיון שכמעט כל פונקציה שתראו בהמשך משתמשת בנתונים השמורים בה.

בראש התוכנית נכתוב את ההגדרה הזו:

```
struct Disc{
   byte Drive;                   // אות הכונן
   word Error;                   // מילת שגיאה
   byte FirstTrack ;             // מספר השיר הראשון
   byte LastTrack ;              // מספר השיר האחרון
   dword LeadOut;                // סוף הדיסק (בתקן LSN)
      } DiscInfo ;
```

השדה הראשון ברשומה זו יכיל את אות הכונן. שאר השדות יוסברו במהלך הפרק.

מהלך ה"שיחה" עם MSCDEX

לאחר שמוודאים שאכן מותקן כונן תקליטורים במחשב, אפשר להתחיל לתקשר עם הדרייבר MSCDEX, או כפי שמקובל לעיתים לומר, "לשוחח" אתו. לשם כך נשתמש בפסיקה 2F עם פרמטר 1510H.

פסיקה זו מצפה לקבל מצביע לבלוק נתונים (תוכן השיחה), אשר תוכנו וגודלו ישתנו על פי הפעולה המבוקשת. קראתי לתהליך מתן ההוראות ל-MSCDEX "שיחה", ולאזור הנתונים - "תוכן השיחה", כי באזור זה המשתמש מכניס נתונים שונים והדרייבר עונה לו (מוסיף נתונים כתגובה לנתונים שהמשתמש הכניס), ממש כמו בשיחה בין שני בני-אדם.

לפני הפעלת הפסיקה, האוגר es יכיל את הסגמנט של בלוק הנתונים, אוגר bx יכיל את ההיסט של בלוק הנתונים בתוך הסגמנט, ואוגר cx יכיל את אות הכונן.

מכיון שאנו רוצים לכתוב פונקציה אחת שתטפל בכל הפניות אל הדרייבר, פונקציה זו צריכה לקבל כפרמטר מצביע מסוג **void**, שלתוכו נכניס את הכתובת של תחילת בלוק הנתונים (תחילת נתוני השיחה).

נפעיל את הפסיקה באמצעות הפונקציה **int86x**, כפי שלמדנו בפרק 3.

הפונקציה תיראה כך:

```
void TalkToMSCDEX ( void *DataBlock )
{
   union REGS ireg;
   struct SREGS sreg;

   ireg.x.ax = 0x1510;                  // פרמטר הפסיקה
   ireg.x.cx = DiscInfo.Drive;          // אות הכונן
   ireg.x.bx = FP_FF (DataBlock);       // ההיסט של בלוק הנתונים
   sreg.es = FP_EG (DataBlock);         // הסגמנט של בלוק הנתונים
   int86x (0x2f,&ireg,&ireg,&sreg);     // הפעלת הפסיקה
}
```

ההבדל היחידי בין "שיחה" עם הדרייבר MSCDEX לבין שיחה רגילה בין בני-אדם היא בכך, שה-MSCDEX (מכיון שהוא דרייבר ולא בן אדם), אינו יודע מתי השיחה מסתיימת. לשם כך, נקבע כי כל שיחה תתחיל במספר משפטים, או למעשה שדות קבועים, שיציגו מה רוצים "לדבר" וכמה זמן השיחה תימשך.

את תוכן השיחה נשמור במבנים. לכל סוג של שיחה דרוש מבנה אחר, המתאים לה, אך כל מבנה שיחה יכיל בתחילתו 5 שדות קבועים ("משפטי פתיחה"):

```
      byte Length ;                // אורך בבתים של בלוק הנתונים
      byte SubUnit ;               // יחידת משנה
      byte Code ;                  // קוד סוג הפקודה
      word Status ;                // סטטוס
      byte Reserved[8] ;           // שמור
```

בשדה הראשון בכל שיחה (שדה **Length**) נאמר ל-MSCDEX מהו אורך השיחה בבתים.

השדה השני בפתיחת השיחה מיועד למחשבים בהם יש Changer, או מגשי תקליטורים, במקום כונן רגיל. התקן מגשי תקליטורים מאפשר להכיל בו יותר מדיסק אחד (במחשב רגיל יש מגירה אחת שבה תקליטור אחד). על כן יש לומר ל-MSCDEX מהי יחידת המשנה, או איזה מבין הדיסקים שבתוך הכונן אנו רוצים להשמיע. ערך זה יהיה מספר בגודל בית, בין אפס למספר הדיסקים שהתקן המגשים (changer) תומך בהם. נכון להיום, התקני מגשים נפוצים במערכות סטריאו ביתיות או במערכות סטריאו לרכב, אך נדירות מאוד במחשבים ביתיים. בשדה **SubUnit** נכניס את הערך אפס.

השדה השלישי בפתיחת השיחה מציין את סוג השיחה. יש חמישה סוגים שונים של שיחות, שלכל אחד מהם קוד שונה (תרשים 14.1). קוד זה נכניס לשדה **Code**.

קוד	סוג שיחה
3	בקשת מידע מה-MSCDEX– **IOCTL INPUT**.
12	מסירת מידע ל-MSCDEX – **IOCTL OUTPUT**.
132	**הפעלת השמעה**.
133	**השהיית השמעה**.
136	**חידוש השמעה** שהופסקה.

תרשים 14.1

השדה הרביעי בפתיחת השיחה, מיועד לקבלת הודעה מ-MSCDEX על ביצוע ההוראה שקיבל, ואם קרתה שגיאה במהלך הביצוע. לכן, את השדה **Status** לא נמלא לפני הקריאה לפונקציה TalkToMSCDEX, והוא יתמלא לבד על ידי MSCDEX. אנו נוכל לבדוק זאת בסיום הפעולה.

את הסטטוס נקבל במשתנה בגודל מילה. ראו את מבנה מילת הסטטוס בתרשים 14.2.

סיבית 15	תכיל אמת אם התרחשה שגיאה.
סיביות 10-14	סיביות שמורות.
סיבית 9	תכיל אמת אם הכונן עסוק בהשמעה.
סיבית 8	תכיל אמת אם הפעולה הסתיימה.
סיביות 0-7	קוד השגיאה (אם סיבית 15 דולקת).

תרשים 14.2

על פי תרשים 14.2 ערך סיבית 15 יהיה אמת אם קרתה שגיאה, לכן נוכל לכתוב פונקציה הבודקת זאת. לדוגמה:

```
int Error (void)
{
   if (DiscInfo.Error&0x8000)
      return(1);
   else return (0);
}
```

על פי תרשים 14.2, ערך סיבית 9 יהיה אמת אם הכונן עסוק בהשמעה. במקרה זה, כל שיחה שנבקש בה מ-MSCDEX, לבצע משהו שדורש הזזת ראש הקריאה, תדליק את סיבית השגיאה.

על פי תרשים 14.2, סיביות 0-7 יכילו את קוד השגיאה אם קרתה שגיאה (ראה רשימת שגיאות בתרשים 14.3).

קוד	תיאור השגיאה
0	לא ניתן לצרוב לדיסק זה.
1	יחידת המשנה אינה מוכרת.
2	הכונן אינו מוכן.
3	פקודה לא מובנת.
4	חל כשל בהעברת השיחה.
5	אורך המבנה אינו נכון.
6	תקלת סריקה.
7	מדיה לא מוכרת.
8	הסקטור לא נמצא.
9	אין דפים במדפסת.
A	שגיאה בצריבה.
B	שגיאה בקריאה.
C	שגיאה כללית (לא ידוע מה הגורם לשגיאה).
D	שמור
E	שמור
F	חילוף דיסקים בלתי חוקי.

תרשים 14.3



כתבו פונקציה המקבלת מילת סטטוס, ומציגה את תיאור התקלה שקרתה. היעזרו בתרשים 14.3 ובפרק 2, "עבודה בסיביות".

השדה החמישי והאחרון בפתיחת השיחה הוא שמור וגודלו שמונה בתים. לכן, לשדה **Reserved** אין צורך להכניס שום ערך.



בכל מקום בפרק זה, בו כתוב **שמור** או **Reserved**, הכוונה היא לכך שהשדה שמור לתפקידים אחרים, שאינם מעניינו של המתכנת, והדבר אינו צריך להטריד אותו. המידע המוסבר בפרק זה מתבסס על המדריך הבסיסי של מיקרוסופט ל-MSCDEX, שהוא המקור הבסיסי ביותר למידע בנושא זה, וכל תוכנה המקשרת עם MSCDEX מתבססת עליו.

לאחר חמשת שדות הפתיחה, מתחילה ה"שיחה" עצמה. על תוכן השיחות המעניינות אותנו (לצורך השמעת תקליטורי שמע), נדון בסעיפים הבאים.

IOCTL INPUT 14.3

בסעיף הקודם נאמר שישנם חמישה סוגי שיחות (תרשים 14.1). בסעיף זה נכיר את הסוג הראשון, הנקרא **IOCTL INPUT**. בשיחות השייכות לסוג זה, אנו מבקשים מידע כלשהו מ-MSCDEX.

לכל שיחות IOCTL יש מבנה שיחה (Data Block) קבוע:

```
struct IOCTL{
      byte Length ;                    // אורך בבתים של בלוק הנתונים
      byte SubUnit ;                   // יחידת משנה
      byte Code ;                      // קוד הפקודה
      word Status ;                    // סטטוס
      byte Reserved[8] ;               // שמור
      byte Reserved2 ;                 // שמור
      dword ControlAddress ;           // כתובת פרמטר הבקרה
      word Bytes ;                     // גודל פרמטר הפקודה
      byte Reserved3[4] ;              // שמור
    };
```

חמשת השדות הראשונים, הם חמשת "משפטי הפתיחה", שהסברנו בסעיף הקודם.

שימו לב! קוד הפקודה הוא 3 (ראו תרשים 14.1).

מבנה זה, הוא המסגרת החיצונית של השיחה, המציג ל-MSCDEX, שאנו רוצים לקבל ממנו מידע. את סוג המידע שאנו מעוניינים בו נשמור במבנה אחר – מבנה הבקרה (ControlBlock), שאת כתובתו נשמור בשדה **ControlAddress** שבמבנה הנתונים, וכך הדרייבר MSCDEX יוכל למצוא אותו.

כדי שיידע מתי מסתיים מבנה הבקרה ויחזור אל מבנה הנתונים (DataBlock), נשמור בשדה **bytes** את גודל מבנה הבקרה, בבתים.

לסיכום, לכל שיחות IOCTL INPUT יש מבנה חיצוני קבוע: מבנה נתונים, DataBlock, הכולל בין השאר את חמשת "משפטי הפתיחה". ואילו החלק העיקרי של השיחה (המשתנה בין כל אחת משיחות ה-IOCTL), נשמר במבנה הבקרה (ControlBlock).

נמשיך במספר דוגמאות לסוג השיחה IOCTL INPUT.

בקשת מידע על התקליטור שבכונן

השיחה הראשונה והחשובה מכולן, היא השיחה בה נבקש מהדרייבר MSCDEX, מידע על אודות התקליטור שנמצא בתוך הכונן.

כך נראה מבנה הבקרה של שיחה זו:

```
struct {
      byte Code
      byte FirstTrack ;
      byte LastTrack ;
      dword LeadOut ;
         } ControlBlock ;
```

בשדה הראשון במבנה הבקרה, נבהיר ל-MSCDEX מהי מטרת השיחה. קוד השיחה לבקשת מידע אודות התקליטור הוא 10, לכן נציב 10 בשדה Code.

ברגע שהדרייבר MSCDEX יזהה קוד זה, הוא יכתוב לתוך מבנה הבקרה שלושה נתונים על אודות התקליטור:

1. מספר רצועת השיר הראשון: **FirstTrack**.
2. מספר רצועת השיר האחרון בתקליטור: **LastTrack**.
3. מיקום הסקטור האחרון בתקליטור: **LeadOut**, על פי התקן של הספר האדום!

עלינו לדעת את מספר רצועת (Track) השיר הראשון בתקליטור, כי תקליטורי מולטימדיה רבים כגון משחקי מחשב שונים, מכילים נתונים ברצועה הראשונה (תוכנת המשחק), ושירים ברצועות שאחריה (מוסיקת הרקע של המשחק). לכן, הרצועה הראשונה בתקליטור אינה בהכרח רצועת השיר הראשון.

בסוף השיחה (לאחר שה-MSCDEX סיים לכתוב את שלושת הנתונים הללו בתוך מבנה הבקרה), נוכל לשמור אותם בתוך המבנה הגלובלי DiscInfo.

שימו לב, שמיקום הסקטור האחרון מוחזר לפי תקן הספר האדום. לכן, כדי שנוכל לבצע חישובים עם ערך זה, עלינו להמירו לערך LSN.

הפונקציה לקבלת מידע אודות התקליטור, תיראה כך:

```
void GetDiscInfo (void)
{
   struct IOCTL DataBlock;              // בלוק הנתונים

   struct {                             // בלוק הבקרה
      byte Code ;                       // קוד הבקרה
      byte FirstTrack ;                 // מספר השיר הראשון
      byte LastTrack ;                  // מספר השיר האחרון
      dword LeadOut ;                   // סוף הדיסק
         } ControlBlock ;

  ///// אתחול מבנה ה-IOCTL בערכים הדרושים לקריאת אינפורמציה על הדיסק /////
   DataBlock.Length = sizeof (DataBlock);  // אורך המבנה
   DataBlock.SubUnit = 0;                  // 0 יחידת משנה
   DataBlock.Code = 3 ;                    // 3 - קריאה
   ControlBlock.Code = 10;                 // 10 - מידע על הדיסק
   DataBlock.ControlAddress = (dword) &ControlBlock;
   DataBlock.Bytes = sizeof (ControlBlock); // גודל הבקרה

  ///// שלחו את המבנה ל-MSCDEX
   TalkToMSCDEX (&DataBlock);
  ///// עדכנו את מבנה הדיסק
   DiscInfo.Error = DataBlock.Status;
   DiscInfo.FirstTrack = ControlBlock.FirstTrack;
   DiscInfo.LastTrack = ControlBlock.LastTrack;
   DiscInfo.LeadOut = ConvertRedBookToLSN (ControlBlock.LeadOut);
}
```

בקשת מידע על השיר

בשיחה זו, נבקש מידע על רצועה (Track) מסוימת. מבנה הבקרה של שיחה זו נראה כך:

```
struct {
      byte Code ;
      byte TrackNumber ;
      dword StartingPoint ;
      byte TrackControl ;
         } ControlBlock ;
```

בשדה הראשון במבנה הבקרה נבהיר ל-MSCDEX מהי מטרת השיחה. קוד השיחה לבקשת מידע אודות רצועה בתקליטור הוא 11, לכן נציב 11 בשדה Code. את מספר הרצועה המבוקשת נציב בשדה **TrackNumber**.

כתגובה, יכתוב MSCDEX לתוך מבנה הבקרה שני פרטי מידע על הרצועה:

1. מיקום תחילת הרצועה (בתקן הספר האדום).
2. מילת בקרה המכילה את מאפייני הרצועה.

בסוף השיחה (לאחר שהדרייבר MSCDEX סיים לכתוב שני נתונים אלה בתוך מבנה הבקרה), נוכל לשמור אותם בתוך מבנה מסוג TrackInfo:

```
struct TrackInfo {
   byte TrackNumber ;                   // מספר השיר
   dword StartingPoint ;                // LSN-מיקום תחילת השיר ב
   byte TrackControl ;                  // בקרת השיר
      } ;
```

שימו לב, שעלינו להמיר את מיקום תחילת השיר מתקן הספר האדום ל-LSN.

מילת הבקרה היא משתנה בגודל בית, אשר כל סיבית בו מספקת תשובה לשאלה כלשהי הנוגעת לרצועה. הסיבית היחידה החשובה לנו היא סיבית מספר 6, העונה על השאלה: האם רצועה זו אינה שיר? לכן, אם ערך הסיבית השישית הוא אמת, לא ננסה להשמיע את הרצועה, כי היא מכילה נתונים ולא שיר.

הפונקציה לשיחה זו, תיראה כך:

```
void GetTrackInfo (struct TrackInfo *Track)
{
   struct IOCTL DataBlock;              // בלוק הנתונים

   struct {                             // בלוק הבקרה
      byte Code ;                       // קוד הבקרה
      byte TrackNumber ;                // מספר השיר המבוקש
      dword StartingPoint ;             // מיקום תחילת השיר
      byte TrackControl ;               // בקרת השיר
         } ControlBlock ;

   ///// אתחול מבנה ה-IOCTL בערכים הדרושים לקריאת מידע על השיר /////
   DataBlock.Length = sizeof (DataBlock);         // אורך המבנה
   DataBlock.SubUnit = 0;                          // 0 יחידת משנה
   DataBlock.Code = 3 ;                            // 3 - קריאה
   ControlBlock.Code = 11;                         // 11 - מידע על שיר
   ControlBlock.TrackNumber=Track->TrackNumber; // מספר השיר
   DataBlock.ControlAddress = (dword) &ControlBlock;
   DataBlock.Bytes = sizeof (ControlBlock);      // גודל הבקרה
```

```
   ///// שלח את המבנה ל-MSCDEX
   TalkToMSCDEX (&DataBlock);

   ///// עדכון הנתונים שהתקבלו
   DiscInfo.Error = DataBlock.Status;
   Track->TrackControl = ControlBlock.TrackControl;
   Track->StartingPoint = ConvertRedBookToLSN (ControlBlock.StartingPoint);
}
```

בקשת מידע על עוצמת הקול

כונן התקליטורים תומך בהשמעת שירים בארבעה ערוצים שונים (השמעה תלת-ממדית).

כל ערוץ מחובר לרמקול אחר באופן הבא:

- ערוץ 0 הוא השמאלי.
- ערוץ 1 הוא הימני.
- ערוץ 2 הוא השמאלי קדמי.
- וערוץ 3 הוא הימני קדמי.

אם כרטיס הקול שבמחשב או התקליטור שמשמיעים, אינם תומכים בארבעה ערוצים, הם יתעלמו מערכי הערוצים 2 ו-3.

את נתוני עוצמת הקול, נשמור במבנה מסוג VolumeInfo, הנראה כך:

```
struct VolumeInfo {
   byte Channel0;
   byte Channel1;
   byte Channel2;
   byte Channel3;
      } ;
```

עוצמת הקול היא ערך בין 0 (ללא קול) ל-255 (עוצמה מקסימלית).

יש אפשרות לכוון את הערוצים, כך שערוץ מסוים של מוסיקה יחובר לערוץ פלט אחר. כך לדוגמה, נוכל להחליף בין הרמקול הימני לבין הרמקול השמאלי.

מבנה הבקרה של השיחה לקבלת מידע על עוצמת הקול נראה כך:

```
struct {
      byte Code ;                   // קוד הבקרה
      byte InputForCh0;             // 0 ערוץ הקלט המחובר לערוץ
      byte Volume0;                 // 0 עוצמת הקול בערוץ פלט
      byte InputForCh1;             // 1 ערוץ הקלט המחובר לערוץ
      byte Volume1;                 // 1 עוצמת הקול בערוץ פלט
      byte InputForCh2;             // 2 ערוץ הקלט המחובר לערוץ
```

```
    byte Volume2;              // 2  עוצמת הקול בערוץ פלט
    byte InputForCh3;          // 3  ערוץ הקלט המחובר לערוץ
    byte Volume3;              // 3  עוצמת הקול בערוץ פלט
       } ControlBlock  ;
```

בשדה הראשון במבנה הבקרה נבהיר ל-MSCDEX מהי מטרת השיחה. קוד השיחה לבקשת מידע אודות עוצמת הקול בערוצים השונים הוא 4, לכן נציב 4 בשדה Code.

כתגובה, הדרייבר יכתוב לתוך מבנה הבקרה שני פרטי מידע על כל אחד מארבעת ערוצי הפלט:

1. מספר ערוץ הקלט המחובר אליו.
2. עוצמת הקול שלו.

בסוף השיחה (לאחר שהדרייבר סיים לכתוב נתונים אלה בתוך מבנה הבקרה), נוכל לשמור אותם בתוך מבנה מסוג VolumeInfo.



כתבו פונקציה בשם GetVolume, המממשת שיחה זו, ומעדכנת רשומה מסוג VolumeInfo בנתוני עוצמת הקול בערוצים.

כזכור, בסוף הפרק תוכלו למצוא את יחידת הספריה המלאה.

בקשת מידע על מצב הכונן

מבנה הבקרה לבקשת מידע על מצב הכונן נראה כך:

```
struct {
      byte Code ;
      dword DeviceControl ;
         } ControlBlock ;
```

בשדה הראשון במבנה הבקרה נבהיר ל-MSCDEX מהי מטרת השיחה. קוד השיחה לבקשת מידע על מצב הכונן הוא 4, לכן נציב 4 בשדה Code.

כתגובה, הדרייבר יכתוב לתוך מבנה הבקרה ערך בגודל מילה כפולה, אשר כל סיבית בו מספקת תשובה לשאלה כלשהי הנוגעת למצב הכונן. מתוך סיביות אלו, שתי סיביות בלבד מעניינות אותנו:

- סיבית 0 העונה על השאלה: האם מגש הכונן פתוח?
- סיבית 1 העונה על השאלה: האם מגש הכונן נעול?

כאשר מגש הכונן נעול, לא ניתן לפתוח אותו באמצעות הלחצן שעל הכונן. אם התוכנית רצה בתוכנת חלונות, ייתכן שהנעילה לא תפעל כראוי כי תוכנת חלונות עוקפת את מנגנון הנעילה.

בסוף השיחה (לאחר שהדרייבר סיים לכתוב את הנתונים האלה בתוך מבנה הבקרה), נוכל לבדוק את ערך סיביות אלו.



כתבו פונקציה בשם GetDeviceInfo, המחזירה את מילת הבקרה של מצב הכונן.



כתבו פונקציות בשם IsTrayOpen ו- IsTrayLocked, הבודקות האם המגש פתוח והאם הוא נעול.

בקשת מידע על מיקום הראש

מבנה הבקרה לבקשת מידע על המיקום שראש הכונן מצביע עליו :

```
struct {
      byte Code ;
      byte Reserved1 ;
      byte TrackNumber ;
      byte Reserved2 ;
      byte MinInTrack ;
      byte SecInTrack ;
      byte FrameInTrack ;
      byte Reserved3 ;
      byte MinInDisc ;
      byte SecInDisc ;
      byte FrameInDisc ;
             } ControlBlock ;
```

בשדה הראשון במבנה הבקרה נבהיר ל-MSCDEX מהי מטרת השיחה. קוד השיחה לבקשת מידע על מיקום הראש הקורא של הכונן הוא 12, לכן נציב 12 בשדה Code.

כתגובה, הדרייבר יכתוב לתוך מבנה הבקרה ערכים שונים, וביניהם מידע על מיקום הראש מתחילת הרצועה הנוכחית, שייכתב בשדות : MinInTrack, SecInTrack ו-FrameInTrack, ומידע על מיקום הראש מתחילת התקליטור, שייכתב בשדות : MinInDisc, SecInDisc ו-FrameInDisc.

בסוף השיחה (לאחר שהדרייבר MSCDEX סיים לכתוב את הנתונים בתוך מבנה הבקרה), נוכל לשמור את המידע הזה במבנה מסוג HeadInfo :

```
struct HeadInfo {
  byte TrackNumber ;          // מספר השיר הנוכחי
  dword LSNt ;                // LSN מתחילת השיר
  dword LSNd ;                // LSN מתחילת הדיסק
     } ;
```



כתבו פונקציה בשם GetHeadInfo, המעדכנת רשומה מסוג HeadInfo בנתוני מיקום הראש.

14.4 IOCTL OUTPUT

בסעיף הקודם הכרנו את סוג השיחות IOCTL **INPUT**, שבהן ביקשנו מידע מהדרייבר MSCDEX. ראינו שכל השיחות השייכות לסוג זה מורכבות ממבנה חיצוני אחיד הזהה לכולם – מבנה הנתונים (DataBlock). בתוכו יש מצביע למבנה פנימי, המותאם אישית לכל אחת מהשיחות – מבנה הבקרה (ControlBlock).

בסעיף זה נכיר את סוג השיחות IOCTL **OUTPUT**. בשיחות מסוג זה נמסור מידע לדרייבר MSCDEX, והוא יבצע על פיו פעולות שונות. גם לשיחות השייכות לסוג זה, יש מבנה חיצוני אחיד – מבנה נתונים אשר דומה לזה של IOCTL INPUT. השוני ביניהם הוא בשדה DataBlock.Code, אשר צריך להכיל כעת את הקוד 12 ולא 3 (ראו תרשים 14.1). מבנה הנתונים, או הבלוק ממומש ב'מבנה' של C הקרוי Struct, כפי שנראה בהמשך.

שליפה או סגירה של המגש

בשיחה זו נמסור לדרייבר MSCDEX, מידע על מצב המגש המבוקש. כתגובה, הוא ישלוף את המגש או יסגור אותו בהתאמה.

בלוק הבקרה של שיחה זו מכיל שדה אחד בלבד המציין את מצב המגש המבוקש. לכן, אין צורך ליצור מבנה מיוחד לשם כך, וניתן לקשר את המשתנה של מצב המגש המבוקש ישירות לשדה ControlAddress שבבלוק הנתונים.

כדי לשלוף מגש - נכניס למשתנה זה את הערך 0, וכדי לסגור מגש - נכניס למשתנה זה את הערך 5. לכן, רצוי להגדיר בראש התוכנית את הקבועים הבאים :

```
#define Eject  0                // פתיחת מגש הכונן
#define Close  5                // סגירת מגש הכונן
```

הפונקציה לקביעת מצב המגש תיראה כך:

```
void Tray (byte Command)
{
   struct IOCTL DataBlock;                        // בלוק הנתונים

   ///// אתחול מבנה ה-IOCTL בערכים הדרושים לטיפול במגש הכונן /////
   DataBlock.Length = sizeof (DataBlock);      // אורך המבנה
   DataBlock.SubUnit = 0;                         // 0 יחידת משנה
   DataBlock.Code = 12 ;                          // 12 - כתיבה
   DataBlock.ControlAddress = (dword)&Command;
                                                  // הבקרה היא הפקודה
   DataBlock.Bytes = 1;                           // 1 גודל הבקרה

   ///// שלחו את המבנה ל-MSCDEX /////
   TalkToMSCDEX (&DataBlock);

   ///// עדכון את הסטטוס /////
   DiscInfo.Error = DataBlock.Status;
}
```

אם כך, כאשר נרצה לשלוף את המגש נכתוב Tray(Eject), וכאשר נרצה לסגור אותו נכתוב Tray(Close).

שחרור או נעילה של מנעול המגש

בשיחה זו נמסור לדרייבר מידע על מצב מנעול המגש המבוקש, כדי שישחרר או ינעל אותו בהתאם.

מבנה הבקרה של שיחה זו נראה כך:

```
struct {
      byte Code ;
      byte Command ;
         } ControlBlock ;
```

בשדה הראשון במבנה הבקרה נמסור ל-MSCDEX מהי מטרת השיחה. קוד השיחה לשחרור/נעילה של מנעול המגש הוא 1, לכן נציב 1 בשדה Code.

בשדה השני (Command) במבנה הבקרה, נמסור ל-MSCDEX מהי הפעולה המבוקשת. כדי לשחרר את מנעול המגש, נכניס לשדה זה את הערך 0 וכדי לנעול את מנעול המגש, נכניס לשדה זה את הערך 1.

לכן, רצוי להגדיר בראש התוכנית את הקבועים הבאים:

```
#define Lock   1                         // נעילת מגש הכונן
#define UnLock 0                         // שחרור מגש הכונן
```

את הפונקציה לשיחה זו נכתוב כך:

```
void TrayLock (byte Command)
{
   struct IOCTL DataBlock;               // בלוק הנתונים

   struct {                              // בלוק הבקרה
      byte Code ;                        // קוד הבקרה
      byte Command ;                     // נעילה/שחרור
         } ControlBlock ;

   ///// אתחול בלוק הנתונים בערכים הדרושים לטיפול במגש הכונן
   DataBlock.Length = sizeof (DataBlock);// אורך המבנה
   DataBlock.SubUnit = 0;                // 0 יחידת משנה
   DataBlock.Code = 12 ;                 // 12 - כתיבה
   ControlBlock.Code = 1;                // 1 - מנעול המגש
   ControlBlock.Command = Command ;       // הפקודה הרצויה
   DataBlock.ControlAddress = (dword)&ControlBlock;
   DataBlock.Bytes = sizeof (ControlBlock); // גודל הבקרה

   ///// MSCDEX-שלחו את המבנה ל
   TalkToMSCDEX (&DataBlock);
   ///// עדכנו את הסטטוס
   DiscInfo.Error = DataBlock.Status;
}
```

קביעת עוצמת הקול

בשיחה זו נמסור לדרייבר מידע על עוצמת הקול הרצויה בערוצים השונים.

מבנה הבקרה של שיחה זו דומה למבנה הבקרה של השיחה לבדיקת עוצמת הקול, אלא שהפעם קוד השיחה יהיה 3.

הפונקציה תיראה כך:

```
void SetVolume (struct VolumeInfo *Vol)
{
   struct IOCTL DataBlock;            // בלוק הנתונים
```

```
  struct {                            // :  בלוק הבקרה
    byte Code ;                       // קוד הבקרה
    byte InputForCh0;                // 0  ערוץ הקלט המחובר לערוץ
    byte Volume0;                    // 0  עוצמת הקול בערוץ פלט
    byte InputForCh1;                // 1  ערוץ הקלט המחובר לערוץ
    byte Volume1;                    // 1  עוצמת הקול בערוץ פלט
    byte InputForCh2;                // 2  ערוץ הקלט המחובר לערוץ
    byte Volume2;                    // 2  עוצמת הקול בערוץ פלט
    byte InputForCh3;               // 3  ערוץ הקלט המחובר לערוץ
    byte Volume3;                   // 3  עוצמת הקול בערוץ פלט
      } ControlBlock ;

  ///// אתחול מבנה ה-IOCTL בערכים הדרושים לקביעת עוצמת הקול /////
  DataBlock.Length = sizeof (DataBlock); // אורך המבנה
  DataBlock.SubUnit = 0;                      // 0 יחידת משנה
  DataBlock.Code = 12 ;                       // 12 - כתיבה
  ControlBlock.Code = 3;                      // 3 - עוצמת הקול
  ControlBlock.InputForCh0 = 0;     // 0 התאמת ערוץ הקלט 0 לפלט
  ControlBlock.InputForCh1 = 1;     // 1 התאמת ערוץ הקלט 1 לפלט
  ControlBlock.InputForCh2 = 2;     // 2 התאמת ערוץ הקלט 2 לפלט
  ControlBlock.InputForCh3 = 3;     // 3 התאמת ערוץ הקלט 3 לפלט
  ControlBlock.Volume0 = Vol->Channel0; // 0 עוצמת הקול בערוץ
  ControlBlock.Volume1 = Vol->Channel1; // 1 עוצמת הקול בערוץ
  ControlBlock.Volume2 = Vol->Channel2; // 2 עוצמת הקול בערוץ
  ControlBlock.Volume3 = Vol->Channel3; // 3 עוצמת הקול בערוץ
  DataBlock.ControlAddress = (dword) &ControlBlock;
  DataBlock.Bytes = sizeof (ControlBlock); // גודל הבקרה

  ///// שלחו את המבנה ל-MSCDEX /////
  TalkToMSCDEX (&DataBlock);

  ///// עדכון הנתונים שהתקבלו /////
  DiscInfo.Error = DataBlock.Status;
}
```



כתבו פונקציה בשם FullVolume, הקובעת את עוצמת הקול בערוצים השונים לערכם המקסימלי.



כתבו פונקציה בשם Mute, המשתיקה את עוצמת הקול בערוצים השונים.

14.5 הפעלת השמעה

השיחה בה נבקש מהדרייבר MSCDEX- להורות לכונן תקליטור לנגן סקטורים מסוימים אינה שייכת ל-IOCTL, ולכן היא בנוייה בדרך שונה. היא אינה מורכבת ממבנה חיצוני ומבנה בקרה פנימי, אלא ממבנה אחד בלבד, בלוק השמעה (PlayBlock), שאינו כולל קריאה למבנה בקרה אחר.

מבנה השיחה נראה כך:

```
struct {
      byte  Length ;
      byte  SubUnit ;
      byte  Code ;
      word  Status ;
      byte  Reserved[8] ;
      byte  Mode ;
      dword StartingSector ;
      dword HowMany ;
         } PlayBlock ;
```

חמשת השדות הראשונים, מייצגים את חמשת "משפטי הפתיחה" הקבועים, כפי שהכרנו במבנה החיצוני של IOCTL. השדה **Code** צריך להכיל את הקוד 132 (על פי תרשים 14.1).

אחרי חמשת משפטי הפתיחה עלינו למסור ל-MSCDEX שלושה ערכים:

1. התקן שאנו משתמשים בו. השדה **Mode** יכיל 0 עבור תקן HSG ו-1 עבור תקן הספר האדום. מכיון שאת מיקום תחילת השיר שמרנו כערך LSN (תקן HSG), נציב בשדה זה אפס.
2. הסקטור הראשון שממנו אנו רוצים להתחיל את ההשמעה (השדה **StartingSector**).
3. כמה סקטורים אנו רוצים שהכונן ינגן (השדה **HowMany**).

אם אנו רוצים להשמיע שיר אחד בלבד, נכניס לשדה StartingPoint את מיקום תחילת רצועת השיר, ולשדה HowMany נכניס את אורך השיר. את אורך השיר נוכל למצוא על ידי חיסור ערך LSN של תחילת השיר מערך LSN של תחילת השיר הבא אחריו.

לשם כך נוכל לכתוב פונקציה הנראית כך:

```
dword TrackLength ( struct TrackInfo Track )
{
   struct TrackInfo NextTrack ;

   if (Track.TrackNumber != DiscInfo.LastTrack)
   {
                           // אם זהו לא סוף השיר, נקרא את נתוני השיר הבא
      NextTrack.TrackNumber = Track.TrackNumber +1 ;
      GetTrackInfo (&NextTrack) ;
```

```
      // אורך השיר הוא ההפרש בין התחלת השיר הבא להתחלת השיר הנוכחי
      return (NextTrack.StartingPoint-Track.StartingPoint);
   }
      // אם זהו סוף הדיסק, נחזיר את ההפרש בין סוף הדיסק לתחילת השיר הנוכחי
   else return (DiscInfo.LeadOut - Track.StartingPoint) ;
}
```

שימו לב! אם זהו השיר האחרון, לא ננסה לקרוא את המידע על הרצועה שאחריו (כי אין כזו), ונחזיר את ההפרש בין סוף הדיסק לתחילת השיר האחרון.

אם ברצוננו להפעיל השמעה שלא תיעצר בסוף השיר, נכניס לשדה HowMany את ההפרש בין סוף הדיסק למיקום, שממנו אנו רוצים שיתחיל לנגן.

אם נכניס לשדה HowMany ערך גדול מדי, לא ייגרם שום נזק. כונן התקליטור עוצר אוטומטית את ההשמעה ברגע שהוא מגיע לסוף הדיסק.

כדוגמה, נכתוב פונקציה המקבלת רשומת שיר, המעודכנת בנתוני הרצועה, ומשתנה נוסף שיכיל אמת, אם ברצוננו שההשמעה תיעצר בתום השיר. הפונקציה נראית כך:

```
void Play (struct TrackInfo *Track, byte OnlyOneTrack)
{
   struct {                             // מבנה בלוק ההשמעה:
      byte  Length ;                    // אורך בבתים של בלוק הנתונים
      byte  SubUnit ;                   // יחידת משנה
      byte  Code ;                      // קוד הפקודה
      word  Status ;                    // סטטוס
      byte  Reserved[8] ;               // שמור
      byte  Mode ;                      // התקן הרצוי
      dword StartingSector ;            // סקטור תחילת השיר
      dword HowMany ;                   // כמה סקטורים לנגן
         } PlayBlock ;

   ///// אתחול בלוק הנתונים בערכים הדרושים להשמעת שיר
   PlayBlock.Length = sizeof (PlayBlock);  // אורך המבנה
   PlayBlock.SubUnit = 0 ;                 // יחידת משנה 0
   PlayBlock.Code = 132 ;                  // קוד 132 - השמעה
   PlayBlock.Mode = 0 ;                    // תקן HSG
   PlayBlock.StartingSector=Track->StartingPoint;
                                           // הסקטור הראשון
   if (OnlyOneTrack)
            // מספר הסקטורים לניגון הוא מספר הסקטורים שמכיל השיר
      PlayBlock.HowMany = TrackLength(*Track);
   else
            // אחרת, נגנו עד סוף הדיסק
      PlayBlock.HowMany = DiscInfo.LeadOut - Track->StartingPoint;
```

```
  ///// MSCDEX-שלח את המבנה ל
  TalkToMSCDEX (&PlayBlock);

  ///// עדכון הנתונים שהתקבלו
  DiscInfo.Error = PlayBlock.Status;
}
```

הערה

ניתן לייעל את התוכנית אם ניצור רשימה מקושרת של רשומות שירים (כמו רשימת האנימציה). בכל פעם שמתחלף הדיסק שבכונן, נקרא לתוך הרשימה את המידע על כל הרצועות שבדיסק, ונחשב לכל אחת מהן את אורכה. בדרך זו, בכל פעם שנרצה להשמיע שיר, לא נצטרך לבקש את המידע על הרצועה הרצויה והרצועה שאחריה, ונוכל לשלוף את הנתונים ישירות מהרשימה. בדרך זו נחסוך בשיחות (חיסכון בזמן), אך נגזול מקום בזיכרון.

14.6 השהיית השמעה

השהיית השמעה (Pause), היא עצירה זמנית של ההשמעה. במצב זה הכונן יפסיק לנגן, הראש יישאר במקומו, וניתן יהיה לחדש את ההשמעה מהנקודה בה ההשמעה נעצרה (על כך נלמד בסעיף 14.6).

לשיחה זו מבנה פשוט. היא מכילה את חמשת "משפטי הפתיחה" בלבד!

```
struct {
      byte  Length ;
      byte  SubUnit ;
      byte  Code ;
      word  Status ;
      byte  Reserved[8] ;
         } PauseBlock ;
```

שימו לב! בסוג שיחה זה, השדה **Code** צריך להכיל את הקוד 133 (על פי תרשים 14.1). הדרייבר אינו זקוק למידע נוסף, לכן שיחה זו מכילה את משפטי הפתיחה בלבד.



כתבו פונקציה בשם Pause המשהה את ההשמעה.

כדי לעצור לחלוטין את ההשמעה (כך שלא נוכל לחדש אותה מהנקודה בה הפסקנו), עלינו להפעיל פעמיים ברציפות את הפונקציה Pause.

ברגע שהדרייבר מקבל שיחה בקוד 133 וההשמעה כבר בהשהיה, הוא מבין שברצוננו לעצור לחלוטין את ההשמעה.

הפונקציה לעצירת ההשמעה תיראה כך:

```
void Stop (void)
{
  Pause();
  Pause();
}
```

14.7 חידוש השמעה

השיחה לחידוש השמעה, הנמצאת בהשהיה, מכילה גם היא את חמשת "משפטי הפתיחה" בלבד, אלא שהפעם קוד השיחה הוא 136.



כתבו פונקציה בשם Resume, המחדשת השמעה הנמצאת בהשהיה.

14.8 יחידה להשמעת תקליטורים

ממשק היחידה

```
/***************************************************************************
 *                          C D  R O M . H                                 *
 *                        ~~~~~~~~~~~~~~~~~~                               *
 *         ממשק ליחידה המתקשרת עם כונן ה- תקליטור בעזרת MSCDEX             *
 ***************************************************************************/

////////////////////////////////////////////////////////////////////////////
//                       ה צ ה ר ו ת                                      //
////////////////////////////////////////////////////////////////////////////

#define Eject  0                          // פתיחת מגש הכונן
#define Close  5                          // סגירת מגש הכונן
#define Lock   1                          // נעילת מגש הכונן
#define UnLock 0                          // שחרור מגש הכונן

typedef unsigned char  byte;              // בית
typedef unsigned short word;              // מילה
typedef unsigned long  dword;             // מילה כפולה
```

```
///// מבנה אשר לתוכו אנו טוענים את המידע הדרוש אודות הדיסק הנוכחי
struct Disc{
   byte Drive;                          // אות הכונן
   word Error;                          // מילת שגיאה
   byte FirstTrack ;                    // מספר השיר הראשון
   byte LastTrack ;                     // מספר השיר האחרון
   dword LeadOut;                       // סוף הדיסק (בתקן LSN)
      } ;

///// מבנה לשמירת מידע אודות עוצמת הקול בערוצים השונים
struct VolumeInfo {
   byte Channel0;                       // עוצמת הקול בערוץ השמאלי
   byte Channel1;                       // עוצמת הקול בערוץ הימני
   byte Channel2;                       // עוצמת הקול בערוץ השמאלי קדמי
   byte Channel3;                       // עוצמת הקול בערוץ הימני קדמי
      } ;
///// מבנה למידע על מיקום ראש הכונן
struct HeadInfo {
   byte TrackNumber ;                   // מספר השיר הנוכחי
   dword LSNt ;                         // LSN מתחילת השיר
   dword LSNd ;                         // LSN מתחילת הדיסק
      } ;
///// מבנה למידע על שיר
struct TrackInfo {
   byte TrackNumber ;                   // מספר השיר
   dword StartingPoint ;                // מיקום תחילת השיר ב-LSN
   byte TrackControl ;                  // בקרת השיר
      } ;
/////  MSCDEX-מבנה IOCTL לקריאה ולכתיבה ל
struct IOCTL{
      byte Length ;                     // אורך בבתים של בלוק הנתונים
      byte SubUnit ;                    // יחידת משנה
      byte Code ;                       // קוד הפקודה
      word Status ;                     // סטטוס
      byte Reserved[8] ;                // שמור
      byte Reserved2 ;                  // שמור
      dword ControlAddress ;            // כתובת פרמטר הבקרה
      word Bytes ;                      // גודל פרמטר הפקודה
      byte Reserved3[4] ;               // שמור
         };
```

```
//////////////////////////////////////////////////////////////////////////
//                          פ ו נ ק צ י ו ת                               //
//////////////////////////////////////////////////////////////////////////

byte CheckMSCDEX (void);
/*------------------------------------------------------------------------
      פונקציה הבודקת האם כונן MSCDEX מותקן כראוי ומזהה את אות ה- תקליטור
      פרמטרים מועברים - אין.
      ערך מוחזר - אמת אם הבדיקה חיובית.
------------------------------------------------------------------------*/

void TalkToMSCDEX ( void *DataBlock );
/*------------------------------------------------------------------------
      פונקציה המתקשרת עם MSCDEX (שולחת לו בלוק נתונים).
      נתונים פרמטרים מועברים - כתובת של בלוק.
      הכתובת נכנסת למצביע מסוג void כדי שהפונקציה תתאים לכל הבלוקים.
      ערך מוחזר - אין.
------------------------------------------------------------------------*/

void TranslateRedBook (dword Value, word *M, byte *S, byte *F);
/*------------------------------------------------------------------------
      פונקציה המתרגמת זמן בתקן הספר האדום (MMMMSSFF) לדקות, שניות ופריימים.
      פרמטרים מועברים :
        Value - הערך שאנו רוצים לתרגם.(בגודל מילה כפולה)
          *M  - דקות (By Address). (בגודל מילה)
          *S  - שניות (By Address). (בגודל בית)
          *F  - פריימים (By Address). (בגודל בית)
      ערך מוחזר - אין.
------------------------------------------------------------------------*/

dword ConvertRedBookToLSN (dword Value);
/*------------------------------------------------------------------------
      פונקציה הממירה זמן בתקן הספר האדום ל-LSN.
      (המספר הלוגי של הסקטור = LSN = Logical Sector Number)
      פרמטרים מועברים - זמן בתקן הספר האדום.
      ערך מוחזר - זמן ב-LSN.
------------------------------------------------------------------------*/
```

```
dword ConvertLSNToRedBook (dword Value);
/*---------------------------------------------------------------------------
                                פונקציה הממירה זמן בתקן LSN ל-ספר האדום.
                (המספר הלוגי של הסקטור = LSN = Logical Sector Number)
                                             פרמטרים מועברים - זמן ב-LSN.
                                              ערך מוחזר - זמן ב-ספר האדום
---------------------------------------------------------------------------*/

void Tray (byte Command);
/*---------------------------------------------------------------------------
                                              פונקציה המטפלת במגש הכונן.
                                  פרמטר מועבר - Command - מה לבצע על הכונן :
                                             Eject  -  שלוף את המגש.
                                             Close  -  סגור את המגש.
                                                       ערך מוחזר - אין.
---------------------------------------------------------------------------*/

void TrayLock (byte Command);
/*---------------------------------------------------------------------------
                                              פונקציה המטפלת במגש הכונן.
                                  פרמטר מועבר - Command - מה לבצע על הכונן :
                                             Lock    -  נעל את המגש.
                                             UnLock  -  שחרר את המגש.
                                                       ערך מוחזר - אין.
---------------------------------------------------------------------------*/

void GetDiscInfo (void);
/*---------------------------------------------------------------------------
 פונקציה השואבת מידע אודות הדיסק הנוכחי שבכונן. (שיר ראשון,אחרון וסוף)
                                                   פרמטרים מועברים - אין.
                                                       ערך מוחזר - אין.
---------------------------------------------------------------------------*/

void GetTrackInfo (struct TrackInfo *Track);
/*---------------------------------------------------------------------------
                                       פונקציה השואבת מידע אודות שיר מסוים.
                           פרמטרים מועברים - כתובת של מבנה מסוג TrackInfo .
   (השדה TrackNumber  צריך להכיל את מספר השיר שעליו ברצוננו לקבל מידע)
                                                       ערך מוחזר - אין.
---------------------------------------------------------------------------*/
```

```
int IsAudioTrack (struct TrackInfo *Track);
/*---------------------------------------------------------------------------
        פונקציה הבודקת האם טראק מסוים הוא מסוג שיר.
        פרמטרים מועברים - כתובת של מבנה מסוג TrackInfo.
        ערך מוחזר - אמת אם זהו שיר.
---------------------------------------------------------------------------*/

dword GetDeviceInfo (void);
/*---------------------------------------------------------------------------
        פונקציה השואבת מידע אודות מצב הכונן.
        פרמטרים מועברים - אין.
        ערך מוחזר - בקרת מצב הכונן (בגודל מילה כפולה).
---------------------------------------------------------------------------*/

int IsTrayOpen (void);
/*---------------------------------------------------------------------------
        פונקציה הבודקת האם מגש הכונן פתוח.
        פרמטרים מועברים - אין.
        ערך מוחזר - אמת אם המגש פתוח.
---------------------------------------------------------------------------*/

int IsTrayLocked (void);
/*---------------------------------------------------------------------------
        פונקציה הבודקת האם מגש הכונן נעול.
        פרמטרים מועברים - אין.
        ערך מוחזר - אמת אם המגש נעול.
---------------------------------------------------------------------------*/

void GetHeadInfo ( struct HeadInfo *Head );
/*---------------------------------------------------------------------------
        פונקציה השואבת מידע אודות מיקום ראש הכונן
        פרמטרים מועברים :
            *Head - כתובת של מבנה מסוג HeadInfo.
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/

int Error (void);
/*---------------------------------------------------------------------------
        פונקציה הבודקת האם התרחשה תקלה        .
        פרמטרים מועברים - אין.
        ערך מוחזר - אמת אם התרחשה תקלה.
--------------------------------------------------------------------------*/
```

```
void GetVolume (struct VolumeInfo *Vol);
/*---------------------------------------------------------------------------
        פונקציה השואבת מידע אודות עוצמת הקול בערוצים השונים.
        פרמטרים מועברים - כתובת של מבנה עוצמת קול.
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/

void SetVolume (struct VolumeInfo *Vol);
/*---------------------------------------------------------------------------
        פונקציה הקובעת את עוצמת הקול בערוצים השונים.
        פרמטרים מועברים - כתובת של מבנה עוצמת קול.
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/

void FullVolume (struct VolumeInfo *Vol);
/*---------------------------------------------------------------------------
        פונקציה הקובעת את עוצמת הקול בערוצים השונים לערכם המקסימלי.
        פרמטרים מועברים - כתובת של מבנה עוצמת קול.
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/

void Mute (struct VolumeInfo *Vol);
/*---------------------------------------------------------------------------
        פונקציה המשתיקה את עוצמת הקול בערוצים השונים.
        פרמטרים מועברים - כתובת של מבנה עוצמת קול.
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/

dword TrackLength ( struct TrackInfo Track );
/*---------------------------------------------------------------------------
        פונקציה המחזירה אורך של שיר ב-LSN.
        פרמטרים מועברים - מבנה (מעודכן) מסוג TrackInfo.
        ערך מוחזר - אורך השיר.
---------------------------------------------------------------------------*/

void Play (struct TrackInfo *Track, byte OnlyOneTrack);
/*---------------------------------------------------------------------------
        פונקציה להשמעה.
        פרמטרים מועברים :
            Track* - כתובת של מבנה (מעודכן) מסוג TrackInfo.
            OnlyOneTrack - האם לנגן רק שיר אחד.
            (אם ערכו 'שקר', ננגן עד לעצירה או סוף הדיסק)
        ערך מוחזר - אין.
---------------------------------------------------------------------------*/
```

```
void Pause (void);
/*---------------------------------------------------------------------------
                                                פונקציה המשהה את ההשמעה.
                                                   פרמטרים מועברים - אין.
                                                         ערך מוחזר - אין.
---------------------------------------------------------------------------*/

void Stop (void);
/*---------------------------------------------------------------------------
                                                פונקציה העוצרת את ההשמעה.
                                                   פרמטרים מועברים - אין.
                                                         ערך מוחזר - אין.
---------------------------------------------------------------------------*/

void Resume (void);
/*---------------------------------------------------------------------------
                                                    פונקציה לביטול השהיה.
                                                   פרמטרים מועברים - אין.
                                                         ערך מוחזר - אין.
---------------------------------------------------------------------------*/
```

מימוש היחידה

```
/****************************************************************************
*                              C D   R O M . C                              *
*             יחידה המתקשרת עם כונן באמצעות תקליטור MSCDEX                   *
****************************************************************************/

/////////////////////////////////////////////////////////////////////////////
//                              ה צ ה ר ו ת                                //
/////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define Eject  0                      // פתיחת מגש הכונן
#define Close  5                      // סגירת מגש הכונן
#define Lock   1                      // נעילת מגש הכונן
#define UnLock 0                      // שחרור מגש הכונן
```

```
typedef unsigned char  byte;              // בית
typedef unsigned short word;              // מילה
typedef unsigned long  dword;             // מילה כפולה

///// מבנה אשר לתוכו אנו טוענים את המידע הדרוש אודות הדיסק הנוכחי
struct Disc{
   byte Drive;                            // אות הכונן
   word Error;                            // מילת שגיאה
   byte FirstTrack ;                      // מספר השיר הראשון
   byte LastTrack ;                       // מספר השיר האחרון
   dword LeadOut;                         // סוף הדיסק (בתקן LSN)
      } DiscInfo ;

///// מבנה לשמירת מידע אודות עוצמת הקול בערוצים השונים
struct VolumeInfo {
   byte Channel0;                         // עוצמת הקול בערוץ השמאלי
   byte Channel1;                         // עוצמת הקול בערוץ הימני
   byte Channel2;                         // עוצמת הקול בערוץ השמאלי קדמי
   byte Channel3;                         // עוצמת הקול בערוץ הימני קדמי
      } ;

///// מבנה למידע על מיקום ראש הכונן
struct HeadInfo {
   byte TrackNumber ;                     // מספר השיר הנוכחי
   dword LSNt ;                           // LSN מתחילת השיר
   dword LSNd ;                           // LSN מתחילת הדיסק
      } ;

///// מבנה למידע על שיר
struct TrackInfo {
   byte TrackNumber ;                     // מספר השיר
   dword StartingPoint ;                  // מיקום תחילת השיר ב-LSN
   byte TrackControl ;                    // בקרת השיר
      } ;

///// מבנה IOCTL לקריאה ולכתיבה ל-MSCDEX
struct IOCTL{
      byte Length ;                       // אורך בבתים של בלוק הנתונים
      byte SubUnit ;                      // יחידת משנה
      byte Code ;                         // קוד הפקודה
      word Status ;                       // סטטוס
      byte Reserved[8] ;                  // שמור
      byte Reserved2 ;                    // שמור
```

```
      dword ControlAddress ;            // כתובת פרמטר הבקרה
      word Bytes ;                      // גודל פרמטר הפקודה
      byte Reserved3[4] ;               // שמור
         };

////////////////////////////////////////////////////////////////////////////
//                          פ ו נ ק צ י ו ת                               //
////////////////////////////////////////////////////////////////////////////

byte CheckMSCDEX (void)
/*------------------------------------------------------------------------
  פונקציה הבודקת האם ה-MSCDEX מותקן כראוי ומזהה את אות כונן ה-תקליטור .
  פרמטרים מועברים - אין.
  ערך מוחזר - אמת אם הבדיקה חיובית.
------------------------------------------------------------------------*/
{
   union REGS ireg ;

   ireg.x.ax = 0x1500;
   int86 (0x2F,&ireg,&ireg);             // הפעלת פסיקה 2fH עם פרמטר 1500

   if (ireg.x.bx)                        // אם ה-MSCDEX מותקן ונמצא כונן
   {
      DiscInfo.Drive =ireg.x.cx ;        // שמור את אות הכונן
      return (1);                        // והחזר אמת
   }
   else return (0);                      // אחרת החזר שקר
}

void TalkToMSCDEX ( void *DataBlock )
/*------------------------------------------------------------------------
  פונקציה המתקשרת עם ה-MSCDEX (שולחת לו בלוק נתונים).
  פרמטרים מועברים - כתובת של בלוק נתונים.
  הכתובת נכנסת למצביע מסוג void כדי שהפונקציה תתאים לכל הבלוקים.
  ערך מוחזר - אין.
------------------------------------------------------------------------*/
{
   union REGS ireg;
   struct SREGS sreg;

   ireg.x.ax = 0x1510;                     // פרמטר הפסיקה
   ireg.x.cx = DiscInfo.Drive;             // אות הכונן
   ireg.x.bx = FP_OFF (DataBlock);         // ההיסט של בלוק הנתונים
```

```
   sreg.es = FP_SEG (DataBlock);          // הסגמנט של בלוק הנתונים
   int86x (0x2f,&ireg,&ireg,&sreg);       // הפעלת הפסיקה
}

void TranslateRedBook (dword Value, word *M, byte *S, byte *F)
/*-----------------------------------------------------------------------------
   פונקציה המתרגמת זמן בתקן הספר האדום (MMMMSSFF) לדקות, שניות ופריימים.
   פרמטרים מועברים :
   Value - הערך שאנו רוצים לתרגם.(בגודל מילה כפולה)
   *M  - דקות (By Address). (בגודל מילה)
   *S  - שניות (By Address). (בגודל בית)
   *F  - פריימים (By Address). (בגודל בית)
   ערך מוחזר - אין.
-----------------------------------------------------------------------------*/
{
  // תקן הספר האדום : MMMMSSFF
  *F =  Value & 0x000000ff;              // פירוק ערך הפריימים
  *S = (Value & 0x0000ff00) >> 8;        // פירוק ערך השניות
  *M = (Value & 0xffff0000) >> 16;       // פירוק ערך הדקות
}

dword ConvertRedBookToLSN (dword Value)
/*-----------------------------------------------------------------------------
   פונקציה הממירה זמן בתקן הספר האדום ל-LSN.
   (המספר הלוגי של הסקטור = LSN = Logical Sector Number)
   פרמטרים מועברים - זמן בתקן הספר האדום.
   ערך מוחזר - זמן ב-LSN.
-----------------------------------------------------------------------------*/
{
  word Min;
  byte Sec, Frame;
  dword LSN;

  TranslateRedBook (Value, &Min, &Sec, &Frame);  //תרגמו את ערך הספר האדום
  LSN = Frame;                       // מספר הפריימים
  LSN +=(dword) Sec * 75;            // + שניות (בשנייה 75 פריימים)
  LSN += (dword) Min * 75 * 60;      // +דקות (75 פריימים בשנייה,60 שניות בדקה)
  LSN -= 150;                        // (תשאלו את ביל גייטס - 150 !!!)

  return (LSN);
}
```

```
dword ConvertLSNToRedBook (dword Value)
/*-------------------------------------------------------------------------
                    פונקציה הממירה זמן בתקן LSN ל-הספר האדום.
                    (המספר הלוגי של הסקטור = LSN = Logical Sector Number)
                    פרמטרים מועברים - זמן ב-LSN.
                    ערך מוחזר - זמן ב-ספר אדום.
-------------------------------------------------------------------------*/
{
  word Min;
  byte Sec, Frame;
  dword Red;

  Value += 150;
  Frame =  Value % 75;          // מספר הפריימים העודפים לאחר חלוקה לשניות
  Value -= Frame;               // החסר את הפריימים העודפים
  Value /= 75;                  // חלוקה לשניות (בשנייה 75 פריימים)
  Sec   =  Value % 60;          // מספר השניות העודפות לאחר חלוקה לדקות
  Value -= Sec;                 // החסר את השניות העודפות
  Min   =  Value / 60;          // חלוקה לדקות

  Red =  (dword) Frame;
  Red += ((dword) Sec) << 8;
  Red += ((dword) Min) << 16;

  return (Red);
}

void Tray (byte Command)
/*-------------------------------------------------------------------------
                    פונקציה המטפלת במגש הכונן.
                    פרמטר מועבר - Command - מה לבצע על הכונן :
                      Eject  -  שלוף את המגש.
                      Close  -  סגור את המגש.
                    ערך מוחזר - אין.
-------------------------------------------------------------------------*/
{
   struct IOCTL DataBlock;                          // בלוק הנתונים
   ///// אתחול מבנה ה-IOCTL בערכים הדרושים לטיפול במגש הכונן
   DataBlock.Length = sizeof (DataBlock);           // אורך המבנה
   DataBlock.SubUnit = 0;                           // יחידת משנה 0
   DataBlock.Code = 12 ;                            // כתיבה - 12
   DataBlock.ControlAddress = (dword)&Command;      // הבקרה היא הפקודה
   DataBlock.Bytes = 1;                             // גודל הבקרה 1
```

```
   ///// שלחו את המבנה ל-MSCDEX
   TalkToMSCDEX (&DataBlock);
   ///// עדכן את הסטטוס
   DiscInfo.Error = DataBlock.Status;
}

void TrayLock (byte Command)
/*----------------------------------------------------------------------------
                                   המטפלת במגש הכונן פונקציה.
                   פרמטר מועבר - Command - מה לבצע על הכונן :
                                   Lock   -  נעל את המגש.
                                   UnLock -  שחרר את המגש.
                                   ערך מוחזר - אין.
----------------------------------------------------------------------------*/
{
   struct IOCTL DataBlock;                          // בלוק הנתונים
   struct {                                         // בלוק הבקרה
      byte Code ;                                   // קוד הבקרה
      byte Command ;                                // נעילה/שחרור
         } ControlBlock ;
   ///// אתחול בלוק הנתונים בערכים הדרושים לטיפול במגש הכונן
   DataBlock.Length = sizeof (DataBlock);           // אורך המבנה
   DataBlock.SubUnit = 0;                           // יחידת משנה 0
   DataBlock.Code = 12 ;                            // כתיבה - 12
   ControlBlock.Code = 1;                           // מנעול המגש - 1
   ControlBlock.Command = Command ;                 // הפקודה הרצויה
   DataBlock.ControlAddress = (dword)&ControlBlock;
   DataBlock.Bytes = sizeof (ControlBlock);         // גודל הבקרה
   ///// שלח את המבנה ל-MSCDEX
   TalkToMSCDEX (&DataBlock);
   ///// עדכן את הסטטוס
   DiscInfo.Error = DataBlock.Status;
}

void GetDiscInfo (void)
/*----------------------------------------------------------------------------
 פונקציה השואבת מידע אודות הדיסק הנוכחי שבכונן. (שיר ראשון,אחרון וסוף)
                                   פרמטרים מועברים - אין.
                                   ערך מוחזר - אין.
----------------------------------------------------------------------------*/
```

```
{
   struct IOCTL DataBlock;                          // בלוק הנתונים
   struct {                                         // בלוק הבקרה
      byte Code ;                                   // קוד הבקרה
      byte FirstTrack ;                             // מספר השיר הראשון
      byte LastTrack ;                              // מספר השיר האחרון
      dword LeadOut ;                               // סוף הדיסק
         } ControlBlock ;
   אתחול מבנה ה-IOCTL בערכים הדרושים לקריאת אינפורמציה על הדיסק /////
   DataBlock.Length = sizeof (DataBlock);           // אורך המבנה
   DataBlock.SubUnit = 0;                           // יחידת משנה 0
   DataBlock.Code = 3 ;                             // קריאה - 3
   ControlBlock.Code = 10;                          // מידע על הדיסק - 10
   DataBlock.ControlAddress = (dword) &ControlBlock;
   DataBlock.Bytes = sizeof (ControlBlock);         // גודל הבקרה
   ///// MSCDEX-שלחו את המבנה ל
   TalkToMSCDEX (&DataBlock);
   ///// עדכנו את מבנה הדיסק
   DiscInfo.Error = DataBlock.Status;
   DiscInfo.FirstTrack = ControlBlock.FirstTrack ;
   DiscInfo.LastTrack = ControlBlock.LastTrack ;
   DiscInfo.LeadOut = ConvertRedBookToLSN(ControlBlock.LeadOut) ;
}

void GetTrackInfo (struct TrackInfo *Track)
/*---------------------------------------------------------------------------
                                   פונקציה השואבת מידע אודות שיר מסוים.
                        פרמטרים מועברים - כתובת של מבנה מסוג TrackInfo .
   (השדה TrackNumber  צריך להכיל את מספר השיר שעליו ברצוננו לקבל מידע)
                                                         ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   struct IOCTL DataBlock;                          // בלוק הנתונים
   struct {                                         // בלוק הבקרה
      byte Code ;                                   // קוד הבקרה
      byte TrackNumber ;                            // מספר השיר המבוקש
      dword StartingPoint ;                         // מיקום תחילת השיר
      byte TrackControl ;                           // בקרת השיר
         } ControlBlock ;
   אתחול מבנה ה-IOCTL בערכים הדרושים לקריאת אינפורמציה על השיר /////
   DataBlock.Length = sizeof (DataBlock);           // אורך המבנה
   DataBlock.SubUnit = 0;                           // יחידת משנה 0
   DataBlock.Code = 3 ;                             // קריאה - 3
```

```
   ControlBlock.Code = 11;                          // מידע על שיר - 11
   ControlBlock.TrackNumber=Track->TrackNumber;  // מספר השיר לבדיקה
   DataBlock.ControlAddress = (dword) &ControlBlock;
   DataBlock.Bytes = sizeof (ControlBlock);        // גודל הבקרה
   ///// שלחו את המבנה ל-MSCDEX
   TalkToMSCDEX (&DataBlock);
   ///// עדכון הנתונים שהתקבלו
   DiscInfo.Error = DataBlock.Status;
   Track->TrackControl = ControlBlock.TrackControl;
      // בבקרה התקבל מיקום הנקודה בפורמט הספר האדום ולכן עלינו להמירו ל-LSN
   Track->StartingPoint = ConvertRedBookToLSN(ControlBlock.StartingPoint);
}

int IsAudioTrack (struct TrackInfo *Track)
/*---------------------------------------------------------------------------
                          פונקציה הבודקת האם טראק מסוים הוא מסוג שיר.
                          פרמטרים מועברים - כתובת של מבנה מסוג TrackInfo.
                          ערך מוחזר - אמת אם זהו שיר.
---------------------------------------------------------------------------*/
{
                          // סיבית מספר 6 במילת הבקרה תכיל אפס אם זהו שיר
   if (Track->TrackControl &0x00000040)
      return(0);
   else return (1);
}

dword GetDeviceInfo (void)
/*---------------------------------------------------------------------------
                          פונקציה השואבת מידע אודות מצב הכונן.
                          פרמטרים מועברים - אין.
                          ערך מוחזר - בקרת מצב הכונן (בגודל מילה כפולה).
---------------------------------------------------------------------------*/
{
   struct IOCTL DataBlock;                          // בלוק הנתונים
   struct {                                         // בלוק הבקרה :
      byte Code ;                                   // קוד הבקרה
      dword DeviceControl ;                         // בקרת הכונן
         } ControlBlock ;
```

```
/////  אתחול מבנה ה-IOCTL בערכים הדרושים לקריאת אינפורמציה על מצב הכונן /////
   DataBlock.Length = sizeof (DataBlock);        // אורך המבנה
   DataBlock.SubUnit = 0;                        // יחידת משנה 0
   DataBlock.Code = 3 ;                          // קריאה - 3
   ControlBlock.Code = 6;                        // מידע על מצב הכונן - 6
   DataBlock.ControlAddress = (dword) &ControlBlock;
   DataBlock.Bytes = sizeof (ControlBlock);      // גודל הבקרה
   ///// שלחו את המבנה ל-MSCDEX
   TalkToMSCDEX (&DataBlock);
   ///// עדכון הנתונים שהתקבלו
   DiscInfo.Error = DataBlock.Status;
   return (ControlBlock.DeviceControl);          // החזר את בקרת מצב הכונן
}

int IsTrayOpen (void)
/*-------------------------------------------------------------------------
                                    פונקציה הבודקת האם מגש הכונן פתוח.
                                    פרמטרים מועברים - אין.
                                    ערך מוחזר - אמת אם המגש פתוח.
--------------------------------------------------------------------------*/
{
   dword Control ;
   Control = GetDeviceInfo ();         // בדוק את מצב הכונן
   if (Control & 0x00000001)           // הסיבית הראשונה מראה את מצב המגש
      return (1);                      // פתוח
   else return (0);                    // סגור
}

int IsTrayLocked (void)
/*-------------------------------------------------------------------------
                                    פונקציה הבודקת האם מגש הכונן נעול.
                                    פרמטרים מועברים - אין.
                                    ערך מוחזר - אמת אם המגש נעול.
--------------------------------------------------------------------------*/
{
   dword Control ;
   Control = GetDeviceInfo ();         // בדוק את מצב הכונן
   if (Control &0x00000002 )           // הסיבית השנייה מראה האם המגש נעול
      return (1);                      // נעול
   else return (0);                    // לא נעול
}
```

```
void GetHeadInfo ( struct HeadInfo *Head )
/*--------------------------------------------------------------------------
                 פונקציה השואבת מידע אודות מיקום ראש הכונן.
                 פרמטרים מועברים :
                   *Head - כתובת של מבנה מסוג HeadInfo.
                 ערך מוחזר - אין.
--------------------------------------------------------------------------*/
{
   dword LSNt , LSNd;                          // LSN מתחילת השיר והדיסק
   struct IOCTL DataBlock;                     // בלוק הנתונים
   struct {                                    // בלוק הבקרה :
      byte Code ;                              // קוד הבקרה
      byte Reserved1 ;                         // שמור
      byte TrackNumber ;                       // מספר הטראק הנוכחי
      byte Reserved2 ;                         // שמור
      byte MinInTrack ;                        // דקות מתחילת השיר
      byte SecInTrack ;                        // שניות מתחילת השיר
      byte FrameInTrack ;                      // פריימים מתחילת השיר
      byte Reserved3 ;                         // שמור
      byte MinInDisc ;                         // דקות מתחילת הדיסק
      byte SecInDisc ;                         // שניות מתחילת הדיסק
      byte FrameInDisc ;                       // פריימים מתחילת הדיסק
         } ControlBlock ;
   ///// אתחול מבנה ה-IOCTL בערכים הדרושים לקריאת אינפורמציה על מיקום הראש
   DataBlock.Length = sizeof (DataBlock);     // אורך המבנה
   DataBlock.SubUnit = 0;                     // יחידת משנה 0
   DataBlock.Code = 3 ;                       // קריאה - 3
   ControlBlock.Code = 12;                    // מידע על מיקום הראש - 12
   DataBlock.ControlAddress = (dword) &ControlBlock;
   DataBlock.Bytes = sizeof (ControlBlock);   // גודל הבקרה
   ///// שלחו את המבנה ל-MSCDEX
   TalkToMSCDEX (&DataBlock);
   ///// עיבוד הנתונים שהתקבלו
   LSNt = ControlBlock.FrameInTrack + (dword)ControlBlock.SecInTrack * 75
       + (dword)ControlBlock.MinInTrack * 75 * 60 -150 ;
   LSNd = ControlBlock.FrameInDisc + (dword)ControlBlock.SecInDisc * 75 +
        (dword)ControlBlock.MinInDisc * 75 * 60 -150 ;
   ///// עדכון הנתונים שהתקבלו
   DiscInfo.Error = DataBlock.Status;
   Head->TrackNumber = ControlBlock.TrackNumber ;
   Head->LSNt = LSNt ;
   Head->LSNd = LSNd ;
}
```

```
int Error (void)
/*--------------------------------------------------------------------------
                                        פונקציה הבודקת האם התרחשה תקלה
                                        פרמטרים מועברים - אין.
                                        ערך מוחזר - אמת אם התרחשה תקלה.
--------------------------------------------------------------------------*/
{
        // בדיקת סיבית מספר 15 בסטטוס הבקרה שמחזירה כל פונקציה לשדה Error
        // ערך הסיבית יהיה אמת אם התרחשה שגיאה
   if (DiscInfo.Error&0x8000)
      return(1);
   else return (0);
}

void GetVolume (struct VolumeInfo *Vol)
/*--------------------------------------------------------------------------
                      פונקציה השואבת מידע אודות עוצמת הקול בערוצים השונים.
                      פרמטרים מועברים - כתובת של מבנה עוצמת קול.
                      ערך מוחזר - אין.
--------------------------------------------------------------------------*/
{
   struct IOCTL DataBlock;                    // בלוק הנתונים
   struct {                                   // בלוק הבקרה :
      byte Code ;                             // קוד הבקרה
      byte InputForCh0;                       // ערוץ הקלט המחובר לערוץ 0
      byte Volume0;                           // עוצמת הקול בערוץ פלט 0
      byte InputForCh1;                       // ערוץ הקלט המחובר לערוץ 1
      byte Volume1;                           // עוצמת הקול בערוץ פלט 1
      byte InputForCh2;                       // ערוץ הקלט המחובר לערוץ 2
      byte Volume2;                           // עוצמת הקול בערוץ פלט 2
      byte InputForCh3;                       // ערוץ הקלט המחובר לערוץ 3
      byte Volume3;                           // עוצמת הקול בערוץ פלט 3
        } ControlBlock  ;
   ///// אתחול מבנה ה-IOCTL בערכים הדרושים לקריאת אינפורמציה על השיר
   DataBlock.Length = sizeof (DataBlock);     // אורך המבנה
   DataBlock.SubUnit = 0;                     // יחידת משנה 0
   DataBlock.Code = 3 ;                       // קריאה - 3
   ControlBlock.Code = 4;                     // מידע על עוצמת הקול - 4
   DataBlock.ControlAddress = (dword) &ControlBlock;
   DataBlock.Bytes = sizeof (ControlBlock);   // גודל הבקרה
   ///// שלחו את המבנה ל-MSCDEX
   TalkToMSCDEX (&DataBlock);
```

```
   ///// עדכון הנתונים שהתקבלו
   DiscInfo.Error = DataBlock.Status;
   Vol->Channel0 = ControlBlock.Volume0;
   Vol->Channel1 = ControlBlock.Volume1;
   Vol->Channel2 = ControlBlock.Volume2;
   Vol->Channel3 = ControlBlock.Volume3;
}

void SetVolume (struct VolumeInfo *Vol)
/*---------------------------------------------------------------------------
                     פונקציה הקובעת את עוצמת הקול בערוצים השונים.
                     פרמטרים מועברים - כתובת של מבנה עוצמת קול.
                     ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   struct IOCTL DataBlock;                    // בלוק הנתונים

   struct {                                   // בלוק הבקרה :
      byte Code ;                             // קוד הבקרה
      byte InputForCh0;                       // ערוץ הקלט המחובר לערוץ 0
      byte Volume0;                           // עוצמת הקול בערוץ פלט 0
      byte InputForCh1;                       // ערוץ הקלט המחובר לערוץ 1
      byte Volume1;                           // עוצמת הקול בערוץ פלט 1
      byte InputForCh2;                       // ערוץ הקלט המחובר לערוץ 2
      byte Volume2;                           // עוצמת הקול בערוץ פלט 2
      byte InputForCh3;                       // ערוץ הקלט המחובר לערוץ 3
      byte Volume3;                           // עוצמת הקול בערוץ פלט 3
         } ControlBlock ;
   ///// אתחול מבנה ה-IOCTL בערכים הדרושים לקביעת עוצמת הקול
   DataBlock.Length = sizeof (DataBlock);     // אורך המבנה
   DataBlock.SubUnit = 0;                     // יחידת משנה 0
   DataBlock.Code = 12 ;                      // כתיבה - 12
   ControlBlock.Code = 3;                     // עוצמת הקול כיוון - 3
   ControlBlock.InputForCh0 = 0;              // התאמת ערוץ הקלט 0 לפלט 0
   ControlBlock.InputForCh1 = 1;              // התאמת ערוץ הקלט 1 לפלט 1
   ControlBlock.InputForCh2 = 2;              // התאמת ערוץ הקלט 2 לפלט 2
   ControlBlock.InputForCh3 = 3;              // התאמת ערוץ הקלט 3 לפלט 3
   ControlBlock.Volume0 = Vol->Channel0;      // עוצמת הקול בערוץ 0
   ControlBlock.Volume1 = Vol->Channel1;      // עוצמת הקול בערוץ 1
   ControlBlock.Volume2 = Vol->Channel2;      // עוצמת הקול בערוץ 2
   ControlBlock.Volume3 = Vol->Channel3;      // עוצמת הקול בערוץ 3
   DataBlock.ControlAddress = (dword) &ControlBlock;
```

```
   DataBlock.Bytes = sizeof (ControlBlock);      // גודל הבקרה
   ///// שלחו את המבנה ל-MSCDEX
   TalkToMSCDEX (&DataBlock);
   ///// עדכון הנתונים שהתקבלו
   DiscInfo.Error = DataBlock.Status;
}

void FullVolume (struct VolumeInfo *Vol)
/*---------------------------------------------------------------------------
       פונקציה הקובעת את עוצמת הקול בערוצים השונים לערכם המקסימלי.
       פרמטרים מועברים - כתובת של מבנה עוצמת קול.
       ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   Vol->Channel0 = 255;
   Vol->Channel1 = 255;
   Vol->Channel2 = 255;
   Vol->Channel3 = 255;
   SetVolume(Vol);
}

void Mute (struct VolumeInfo *Vol)
/*---------------------------------------------------------------------------
       פונקציה המשתיקה את עוצמת הקול בערוצים השונים.
       פרמטרים מועברים - כתובת של מבנה עוצמת קול.
       ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   Vol->Channel0 = 0;
   Vol->Channel1 = 0;
   Vol->Channel2 = 0;
   Vol->Channel3 = 0;
   SetVolume(Vol);
}

dword TrackLength ( struct TrackInfo Track )
/*---------------------------------------------------------------------------
       פונקציה המחזירה אורך של שיר ב-LSN.
       פרמטרים מועברים - מבנה (מעודכן) מסוג TrackInfo.
       ערך מוחזר - אורך השיר.
---------------------------------------------------------------------------*/
```

```
{
   struct TrackInfo NextTrack ;
   if (Track.TrackNumber != DiscInfo.LastTrack)
   {
                        // אם זהו לא סוף השיר, קרא את נתוני השיר הבא
      NextTrack.TrackNumber = Track.TrackNumber +1 ;
      GetTrackInfo (&NextTrack) ;
               // אורך השיר הוא ההפרש בין התחלת השיר הבא להתחלת השיר הנוכחי
      return (NextTrack.StartingPoint-Track.StartingPoint);
   }
       // אם זהו סוף הדיסק, החזר את ההפרש בין סוף הדיסק לתחילת השיר הנוכחי
   else return (DiscInfo.LeadOut - Track.StartingPoint) ;
}

void Play (struct TrackInfo *Track, byte OnlyOneTrack)
/*--------------------------------------------------------------------------
                                                     פונקציה להשמעה.
                                                     פרמטרים מועברים :
                 Track* - כתובת של מבנה (מעודכן) מסוג TrackInfo.
                               OnlyOneTrack - האם לנגן רק שיר אחד.
      (אם ערכו 'שקר', נגן עד לעצירה או סוף הדיסק)
                                                     ערך מוחזר - אין.
--------------------------------------------------------------------------*/
{
   struct {                              // מבנה בלוק ההשמעה:
      byte  Length ;                     // אורך בבתים של בלוק הנתונים
      byte  SubUnit ;                    // יחידת משנה
      byte  Code ;                       // קוד הפקודה
      word  Status ;                     // סטטוס
      byte  Reserved[8] ;                // שמור
      byte  Mode ;                       // התקן הרצוי
      dword StartingSector ;             // סקטור תחילת השיר
      dword HowMany ;                    // כמה סקטורים לנגן
         } PlayBlock ;

   ///// אתחול בלוק הנתונים בערכים הדרושים להשמעת שיר
   PlayBlock.Length = sizeof (PlayBlock);        // אורך המבנה
   PlayBlock.SubUnit = 0 ;                       // יחידת משנה 0
   PlayBlock.Code = 132 ;                        // קוד 132 - השמעה
   PlayBlock.Mode = 0 ;                          // תקן HSG
   PlayBlock.StartingSector=Track->StartingPoint;// הסקטור הראשון
```

```
   if (OnlyOneTrack)
                      מספר הסקטורים לניגון הוא מספר הסקטורים שמכיל השיר //
      PlayBlock.HowMany = TrackLength(*Track);
   else
                      אחרת,נגן עד סוף הדיסק //
      PlayBlock.HowMany = DiscInfo.LeadOut - Track->StartingPoint;
   שלח את המבנה ל-MSCDEX /////
   TalkToMSCDEX (&PlayBlock);
   עדכון הנתונים שהתקבלו /////
   DiscInfo.Error = PlayBlock.Status;
}
void Pause (void)
/*---------------------------------------------------------------------------
                                              פונקציה המשהה את ההשמעה.
                                              פרמטרים מועברים - אין.
                                              ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   struct {                              מבנה בלוק השהיה: //
      byte  Length ;                     אורך בבתים של בלוק הנתונים //
      byte  SubUnit ;                    יחידת משנה //
      byte  Code ;                       קוד הפקודה //
      word  Status ;                     סטטוס //
      byte  Reserved[8] ;                שמור //
         } PauseBlock ;
   אתחול בלוק הנתונים בערכים הדרושים להשהיה /////
   PauseBlock.Length = sizeof (PauseBlock);        אורך המבנה //
   PauseBlock.SubUnit = 0 ;                        יחידת משנה 0 //
   PauseBlock.Code = 133 ;                         קוד 133 - השהיה //
   שלח את המבנה ל-MSCDEX /////
   TalkToMSCDEX (&PauseBlock);
   עדכון הנתונים שהתקבלו /////
   DiscInfo.Error = PauseBlock.Status;
}

void Stop (void)
/*---------------------------------------------------------------------------
                                              פונקציה העוצרת את ההשמעה.
                                              פרמטרים מועברים - אין.
                                              ערך מוחזר - אין.
---------------------------------------------------------------------------*/
```

```
{
   ///// כאשר מפעילים פעמיים ברציפות את פקודת ההשהיה ההשמעה נעצרת לחלוטין
   Pause();
   Pause();
}

void Resume (void)
/*---------------------------------------------------------------------------
                                                    פונקציה לביטול השהיה.
                                                 פרמטרים מועברים - אין.
                                                       ערך מוחזר - אין.
---------------------------------------------------------------------------*/
{
   struct {                              // :מבנה בלוק ביטול השהיה
      byte  Length ;                     // אורך בבתים של בלוק הנתונים
      byte  SubUnit ;                    // יחידת משנה
      byte  Code ;                       // קוד הפקודה
      word  Status ;                     // סטטוס
      byte  Reserved[8] ;                // שמור
         } ResumeBlock ;
   ///// אתחול בלוק הנתונים בערכים הדרושים לביטול ההשהיה
   ResumeBlock.Length = sizeof (ResumeBlock);   // אורך המבנה
   ResumeBlock.SubUnit = 0 ;                    // יחידת משנה 0
   ResumeBlock.Code = 136 ;                     // קוד 136 - ביטול השהיה
   ///// MSCDEX-שלח את המבנה ל
   TalkToMSCDEX (&ResumeBlock);
   ///// עדכון הנתונים שהתקבלו
   DiscInfo.Error = ResumeBlock.Status;
}
```

תוכנית לדוגמה

התוכנית לדוגמה של פרק זה Compact.c, שמצורפת בדיסק (תחת התיקיה books\59281), מראה כיצד ניתן לבנות נגן תקליטורים בסיסי בעזרת היחידה שבנינו.

הקבצים שיש לצרף לפרויקט כדי להריץ את הדוגמה הם: Mytime.c ,Viewbmp.c ,Vga256.c, Gmouse.c ,Cdrom.c ו-Compact.c. למידע נוסף קראו את נספח ו'.



תרשים 14.4: לכידת המסך בזמן הרצת תוכנית הדוגמה



נסו להרחיב את נגן התקליטורים, כך שיוכל לבצע את הדברים הבאים:

- קביעת רשימת שירים להשמעה (Play List).
- אפשרות להשמעת שירים באופן אקראי.
- אפשרות להשמעה מחזורית (ללא הפסקה).

פרק 15

השמעת קבצי WAV

בפרק זה נלמד להשמיע קבצי קול מסוג WAV.

- ✔ **נלמד לזהות כתובות ומאפייני כרטיס הקול.**
- ✔ **נלמד כיצד בנוי קובץ WAV.**
- ✔ **נלמד כיצד פועל מנגנון ההשמעה בעזרת הפסיקות.**
- ✔ **נלמד לשלוט על עוצמת הקול ואיזון הערוצים.**
- ✔ **לסיום, נלמד להפעיל ולעצור השמעת קבצי WAV.**

הערה

מומלץ לחזור על פרק 2, "עבודה בסיביות", לפני קריאת פרק זה.

15.1 זיהוי מאפייני כרטיס הקול

כדי שנוכל להשתמש בכרטיס קול, עלינו לדעת עליו שלושה מאפיינים חשובים:

1. מהי כתובת הבסיס של ה-DSP (Digital Signal Processor – מעבד האותות הדיגיטליים).
2. מהו ערוץ בקשת הפסיקה – **IRQ Level** (Interrupt ReQuest).
3. מהו ערוץ הגישה הישירה לזיכרון – **DMA Channel** (Direct Memory Access).

אם שיחקתם בעבר במשחקים שפעלו ב-DOS, ודאי נתקלתם בשלושה מושגים אלה, ונתבקשתם למלא אותם בעת התקנת המשחק, כדי שיוכל לפעול עם כרטיס הקול שמותקן במחשב.

אנחנו ננסה לזהות מאפיינים אלה, בלי לבקש מהמשתמש להקליד אותם בעת ההתקנה.

הדרך הפשוטה ביותר לזהות את ה-IRQ וה-DMA, היא לקרוא את המחרוזת המתארת את סביבת העבודה (**environment**) של כרטיס הקול, ולחלץ ממנה את הנתונים הרצויים.

נסו להקליד את הפקודה **set** במערכת ההפעלה שלכם. אם אתם עובדים תחת מערכת הפעלה מסוג Windows, פתחו חלון DOS (DOS Command) והקישו בו את הפקודה. לאחר

הקשת הפקודה, יופיעו על המסך מספר שורות, המכילות את מחרוזות משתני הסביבה של מערכת ההפעלה Windows (Environment). ראו דוגמה בתרשים 15.1.

```
Microsoft(R) Windows 98
Copyright Microsoft Corp 1981-1999.©

C:\WINDOWS>set
TMP=C:\WINDOWS\TEMP
TEMP=C:\WINDOWS\TEMP
PROMPT=$p$g
winbootdir=C:\WINDOWS
PATH=C:\WINDOWS;C:\WINDOWS\COMMAND
COMSPEC=C:\WINDOWS\COMMAND.COM
windir=C:\WINDOWS
BLASTER=A220 I2 D1 H5 P330 T6

C:\WINDOWS>
```

תרשים 15.1

אם אתם קוראים פרק זה, אני מניח שיש במחשב שלכם כרטיס קול (רכיב על לוח האם, או כרטיס נפרד המורכב בחריץ). על כן, בתחילת אחת השורות האלו, אמורה להופיע המילה **BLASTER**. שורה זו היא מחרוזת משתני הסביבה של כרטיס הקול. במחרוזת זו מופיע לאחר סימן השוויון, רצף של ערכים ואותיות, המייצגים נתונים שונים אודות כרטיס הקול. סדר הערכים וגודל האותיות (רישיות או רגילות) אינו קבוע, אך תמיד חייבת להופיע האות I (או i) ואחריה ספרה אחת או שתיים, המייצגת את מספר בקשת הפסיקה (IRQ), וגם האות D (או d) ואחריה ספרה אחת, המייצגת את מספר ערוץ הגישה הישירה (DMA).

תחילה, עלינו להגדיר מבנה גלובלי שלתוכו נכניס את הערכים הדרושים:

```
struct {                          // מבנה לשמירת הנתונים הנחוצים אודות כרטיס הקול
   char DMA ;                     // מספר ערוץ הגישה הישירה לזיכרון
   char IRQ ;                     // מספר בקשת הפסיקה
   short Base ;                   // Sound Blaster -כתובת הבסיס של ה
      } SoundBlaster ;
```

כדי לקלוט את המחרוזת הזאת לתוך התוכנית, נשתמש בפונקציית הספריה **getenv**, המקבלת מחרוזת ומחזירה את מחרוזת ה-environment של אותה מחרוזת (אם היא קיימת).

```
char *BLASTER;
BLASTER = getenv ("BLASTER");
```

לאחר קליטת המחרוזת המתארת את סביבת העבודה של כרטיס הקול, עלינו לחפש בה את התו 'D' או 'd', להמיר את התו שאחריו לספרה ולהציב את הספרה במבנה שבנינו.

```
for (i = 0; i < strlen (BLASTER); i++)
   if ((BLASTER [i] | 32) == 'd')              // D או d
      SoundBlaster.DMA = BLASTER [i + 1] - '0';
```

כך גם נעשה לאות I, אלא שהפעם נתחשב בעובדה, שהערך המבוקש יכול להיות בעל שתי ספרות:

```
for (i = 0; i < strlen (BLASTER); i++)
   if ((BLASTER [i] | 32) == 'i')
   {
      SoundBlaster.IRQ = BLASTER [i + 1] - '0';
      if (BLASTER [i + 2] != ' ')           // הערך הוא דו ספרתי
      {
         SoundBlaster.IRQ *= 10 ;
         SoundBlaster.IRQ += BLASTER[i + 2] - '0';
      }
   }
```

את כתובת הבסיס של ה-DSP (מעבד האותות הדיגיטליים - רכיב חומרה שנמצא על כרטיס הקול ומהווה חלק ממנו, הסבר מפורט יופיע בהמשך הפרק), נמצא בדרך אחרת. כתובת זו צריכה להיות ערך בין 210H ל-280H המתחלק ב-10H. כלומר, עלינו לבדוק מקסימום שמונה כתובות. כדי לבדוק האם כתובת כלשהי היא כתובת הבסיס האמיתית, עלינו לבצע את הפעולות הבאות:

1. לשלוח ליציאה של המחשב (port), שבכתובת הנבדקת בהיסט של 6H את הערך 1.
2. לאחר מאית שנייה, לשלוח ליציאה שבכתובת הנבדקת בהיסט של 6H את הערך 0.
3. לקרוא ערך מהיציאה שבכתובת הנבדקת בהיסט של eH, ולוודא שערך הסיבית האחרונה (b7) הוא "1" לוגי.
4. לקרוא ערך מהיציאה שבכתובת הנבדקת בהיסט של aH, ולוודא שערכו הוא aaH.

אם אחד משני התנאים (שבפעולות 3 ו-4) לא התקיים, אין זו כתובת הבסיס של מעבד האותות הדיגיטליים (ה-DSP).

פונקציית הבדיקה תיראה כך:

```
int CheckBaseAddress(short AddressToCheck)
{
   outportb (AddressToCheck+0x6,1);
   delay (10);
   outportb (AddressToCheck+0x6,0);
   delay (10);

   if (  ( ( inportb(AddressToCheck+0xE)&0x80 ) == 0x80 ) &&
         ( inportb(AddressToCheck+0xA) == 0xAA )  )
   {
      ///// אם הכתובת שנבדקה היא אכן כתובת הבסיס של ה-DSP
      SoundBlaster.Base = AddressToCheck ;
      return (1) ;
   }
return 0;
}
```

עכשיו נפעיל את פונקציית הבדיקה על כל אחת משמונה הכתובות האפשריות, עד שנמצא את כתובת הבסיס האמיתית.

```
   char i=1;
   int Found = 0 ;
   SoundBlaster.Base = 0;                    //עדיין לא נמצאה שום כתובת

   /////  חיפוש כתובת ה- SoundBlaster
   while (i<9 && !Found)
   {
      if (CheckBaseAddress (0x200 + (i << 4)))  // קפיצות של 10H
         Found = 1;
      i++;
   }
   if (i == 9)
      return(0);                         // אם לא נמצא, החזר "שקר"
```

DSP - מעבד אותות דיגיטליים

DSP (Digital Signal Processor), הוא רכיב חומרה – מעבד אותות דיגיטליים – שנמצא בכרטיס הקול. תפעול כרטיס הקול מתבצע על ידי שליחת פקודות שונות (שנכיר בהמשך הפרק) אל המעבד הזה. לאחר שזיהינו (בסעיף הקודם) את כתובת הבסיס של ה- DSP, נוכל לשלוח לו פקודות.

נעשה זאת כך:

תחילה, עלינו להמתין עד שמעבד האותות הדיגיטליים (DSP) יהיה זמין. כל עוד הוא אינו פנוי, כאשר נקרא ערך מהיציאה שבכתובת הבסיס בהיסט של cH, ערך הסיבית האחרונה בו (b7) יהיה אחד, לכן נחכה עד שיהפוך לאפס.

ברגע שמעבד האותות זמין, ניתן לשלוח אליו את הפקודה הרצויה, על ידי שליחת ערך הפקודה ליציאה שבכתובת הבסיס בהיסט של eH.

הפונקציה לכתיבת פקודה למעבד האותות DSP, תיראה כך:

```
void WriteDSP(byte Value)
{
                                    // המתן שה-DSP יהיה זמין
   while ((inportb (SoundBlaster.Base + 0xC) & 0x80) == 0x80);

                                    // שליחת הערך ל-DSP
   outportb (SoundBlaster.Base + 0xC, Value);
}
```

הערה

הערה חשובה! אם אתם עובדים תחת DOS או Windows 95, ניתן להפעיל את התוכניות בהקשה כפולה על קובץ EXE שתצרו. אך אם אתם עובדים עם Windows 98 ומעלה, יש להפעיל את התוכנית מתוך חלון DOS (DOS Command), כדי שהזיהוי האוטומטי יפעל כראוי. הסיבה לכך נעוצה בעובדה שהחל מ- Windows 98, הגישה למחרוזות environment מוגבלת יותר, וניתן להגיע אליהן רק מתוך חלון DOS.

אם כך, עומדות לפנינו שתי אפשרויות:

1. במדריך למשתמש של הפרויקט, נכתוב הערה המסבירה, שיש להריץ אותו מתוך חלון DOS.
2. נאפשר למשתמש להקליד בעצמו את שלושת מאפייני כרטיס הקול, במידה והזיהוי האוטומטי לא הצליח. אם במשחקים כמו Doom ו-Duke זה היה כך, גם אצלכם זה יכול להיות כך.

15.2 קבצי WAV

WAV הוא הפורמט הנפוץ ביותר לקבצי קול. היתרון שבו הוא בכך, שערכי הצלילים שהוא מייצג אינם דחוסים (כמו לדוגמה בפורמט MP3), והדבר מקל על המתכנתים בהשמעתו ובהקלטתו. עם זאת, יתרון זה בא על חשבון נפח הקובץ. למרות העובדה, שנפח קבצי WAV גדול מהפורמטים הדחוסים, הוא עדיין נפוץ להשמעת אפקטים וצלילים קצרים.

ישנם סוגים שונים של קבצי WAV. הסוגים השונים נבדלים ב-3 מאפיינים עיקריים, והם:

1. מספר הערוצים (מונו או סטריאו).
2. רזולוציית הסיביות – **Bit Resolution** (מספר הסיביות שמייצגות כל דגימה).
3. קצב הדגימה – **Sample Rate** (גם תדר דגימה: מספר הדגימות בשנייה).

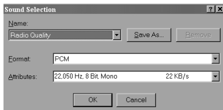
יחידת הספריה, המוסברת בפרק זה, תומכת בקבצי WAV מסוג 8bit PCM Mono בכל התדירויות. אמנם האיכות נמוכה, אך נפח הקבצים קטן מאוד יחסית לשאר הסוגים. הנפח הקטן מאפשר אחסון של קבצי קול רבים בדיסקט בודד, ולכן הוא מתאים ביותר לפרויקטים קטנים בהיקפם.

כאשר צריך לעבוד עם קבצים באיכות גבוהה יותר, אינכם צריכים לחשוש, כי יש לכך פתרון. עד סוף הפרק יהיה בידכם הידע, לכתוב יחידה התומכת בכל סוג WAV שתרצו.

כיצד נמיר קבצי WAV כדי שיתאימו לפורמט הרצוי?

אם יש בידכם קבצי WAV שאינם מסוג 8bit מונו, אל דאגה, כיון שניתן להמירם בקלות. כל שעליכם לעשות הוא:

1. פתחו את הקובץ בעזרת תוכנת הרשמקול (Sound Recorder) של Windows.
אם לא התקנתם תוכנה אחרת להשמעת קבצי WAV, זו ברירת המחדל ללחיצה כפולה על קובץ WAV. אם התקנתם תוכנה אחרת, תוכלו להגיע לרשמקול על ידי תפריט התחל>תוכניות>עזרים>בידור.
2. לחצו על תפריט קובץ>מאפיינים. ייפתח חלון ובמרכזו לחצן שכתוב עליו המר כעת (Convert now), לחצו עליו.
3. בתיבת הבחירה העליונה - שם (Name), בחרו איכות רדיו (Radio Quality), ולחצו על לחצן אישור (OK).



תרשים 15.2

15.3 מבנה הקובץ

קובץ WAV מורכב מבלוקים של נתונים (Chunks). לפני כל בלוק נמצא קוד הזיהוי שלו (בגודל 4 בתים), ושדה המכיל את גודל הבלוק בבתים (גם גודלו 4 בתים).
מתוך הבלוקים שבקובץ, אנו מעוניינים בשלושה מהם:

1. בלוק ה-RIFF.
2. בלוק פורמט קובץ ה-WAV.
3. בלוק נתוני הצלילים.

בלוק ה-RIFF, הוא הבלוק הראשון בקובץ, ומספר הזיהוי שלו הוא 46464952H. בכל קובץ WAV תקין, ארבעת הבתים הראשונים בבלוק RIFF, צריכים להכיל את הערך 45564157H. אם כך, נוכל לבדוק האם הקובץ הוא קובץ WAV תקין. נגדיר תחילה מבנה שיכיל את תחילת בלוק RIFF:

```
struct {
   long BlockType; // (0x46464952 - קוד הזיהוי של הבלוק (צ"ל
   long BlockSize; // גודל הבלוק
   long RIFF;      // (0x45564157 - ערך RIFF (בקובץ WAV תקין צ"ל
  } RIFFBlock;
```

כעת נוכל לבדוק את תקינות הקובץ:

```
   WAVFile = fopen(FileName, "rb");
   fread (&RIFFBlock, sizeof (RIFFBlock), 1, WAVFile);
   if ( RIFFBlock.BlockType != 0x46464952 ||
       RIFFBlock.RIFF != 0x45564157)     //אם זהו קובץ WAV שאינו תקין
   {
      fclose (WAVFile);
      return(0);
   }
```

הבלוק השני המעניין אותנו, הוא בלוק המתאר את פורמט WAV. קוד הזיהוי שלו הוא 20746d66H. הבלוק נראה כך:

```
struct {                        // בלוק פורמט קובץ WAV
   long BlockType;              // (0x20746D66 - קוד הזיהוי של הבלוק (צ"ל
   long BlockSize;              // גודל הבלוק
   short WaveType;              // סוג קובץ WAV
   short Channels;              // מספר ערוצי השמע
   long SampleRate;             // קצב הדגימה
   long BytesPerSecond;         // (BPS) בתים לשנייה
   short BlockAlignment;        // יישור הסיביות בבלוק הנתונים
   short BitResolution;         // רזולוציית הסיביות
  } WAVFormatBlock;
```

כדי למצוא בלוק זה, עלינו לקרוא מהקובץ לתוך המבנה שהגדרנו, ולבדוק האם שדה קוד זיהוי הבלוק (BlockType) הוא אכן 20746d66H. אם אין זה הבלוק שאנו מצפים לו, קרא את תחילתו של בלוק אחר, דלגו לתחילת הבלוק הבא ונסו שוב ושוב עד שתמצאו את הבלוק הרצוי.

כיצד נדע היכן מתחיל הבלוק הבא? לשם כך, נשתמש בשדה השני בכל בלוק – שדה גודל הבלוק. נעשה זאת כך:

לפני שנקרא בלוק מהקובץ, נשמור את המיקום הנוכחי שלנו בקובץ במשתנה זמני. אם לאחר קריאת המבנה נגלה שקראנו את תחילתו של בלוק שאינו זה שאנו מחפשים, נקפוץ מתחילת הקובץ (SET_SEEK) לנקודה הנמצאת במרחק, שיחושב כך: נחבר את המרחק מתחילת הקובץ עד לתחילת הבלוק הנוכחי (המרחק ששמרנו במשתנה הזמני) ואת גודל הבלוק (שנשמר בשדה BlockSize). לסיום, נוסיף שמונה בתים.

תזכורת: עלינו להוסיף שמונה בתים, מכיון ששני השדות הראשונים (קוד הזיהוי וגודל הבלוק), שכל אחד מהם בגודל ארבעה בתים, נחשבים כהקדמה לקובץ וגודלם אינו נכלל בערך הרשום בשדה גודל הבלוק.

נבצע כך את הקריאה:

```
long temp;

do
   {
      Temp = ftell (WAVFile);
      fread (&WAVFormatBlock,sizeof(WAVFormatBlock),1,
         WAVFile);
      fseek (WAVFile,
         Temp+WAVFormatBlock.BlockSize+8,SEEK_SET);
   } while (WAVFormatBlock.BlockType != 0x20746D66);
```

עכשיו יש בידינו כל המידע הדרוש על סוג הקובץ. לפני שנוכל להתחיל בהשמעה, עלינו למסור למעבד האותות (DSP), באיזה תדר אנו רוצים שיפעל. נעשה זאת באמצעות הפקודה 40H, שפועלת כך: תחילה אנו שולחים ל-DSP את קוד הפקודה (40H), ולאחר מכן – את תוצאת החלוקה של 1000000 בקצב הדגימה (Sample Rate). התוצאה המתקבלת מחוסרת מ-256. הפקודות הדרושות נראות כך:

```
   ///// קביעת קצב הדגימה
   WriteDSP (0x40);
   WriteDSP (256 - 1000000 / WAVFormatBlock.SampleRate);
```

הבלוק השלישי המעניין אותנו בקובץ, הוא הבלוק העיקרי המכיל את ערכי הצלילים. קוד הזיהוי שלו הוא 61746164H. נגדיר מבנה, שלתוכו נכניס את ההקדמה של בלוק נתוני הצלילים:

```
struct {                        // הנתונים להשמעה  תחילת בלוק
   long BlockType;              // (0x61746164 - קוד הזיהוי של הבלוק (צ"ל
   long BlockSize;              // גודל הבלוק
       } BeginingOfDataBlock;
```

כדי למצוא בלוק זה, נשתמש בשיטה, שהשתמשנו בה למציאת הבלוק הקודם:

```
do
   {
      Temp = ftell (WAVFile);
      fread (&BeginingOfDataBlock,sizeof(
         BeginingOfDataBlock),1,WAVFile);
      if (BeginingOfDataBlock.BlockType != 0x61746164)
        fseek (WAVFile,Temp+BeginingOfDataBlock.
       BlockSize+8,SEEK_SET);
   } while (BeginingOfDataBlock.BlockType != 0x61746164);
```

15.4 מנגנון ההשמעה המחזורית

האם קרה לכם אי פעם ששיחקתם באיזשהו משחק, ולפתע המחשב נתקע? אם למשחק היתה מוסיקת רקע, שמתם לב בוודאי, שיש מקרים שבהם המשחק תקוע והמקשים אינם מגיבים, אך המוסיקה ממשיכה לנגן כאילו הכל בסדר. נסו לחשוב כיצד זה ייתכן.

הסיבה לכך נעוצה בעובדה, שמנגנון ההשמעה פועל בעזרת פסיקות בלבד. כלומר:

1. ההשמעה לא תפריע למהלך התוכנית כלל! (וזהו יתרון נהדר).
2. כיון שרמת פסיקות החומרה גבוהה מרמת התוכנה, גם אם היישום (הפועל ברמת התוכנה) יפסיק לפעול, ההשמעה (הפועלת ברמת פסיקות החומרה) תימשך.

בכל מחזור, ה-DSP מנגן 16KB ואחר כך מפעיל פסיקה, הקוראת מחוצץ DMA (שגודלו 32KB) את 16KB הבאים (חצי מהחוצץ), הישר ל-DSP (שמתחיל מיידית להשמיע אותם). אחר כך, הפסיקה קוראת מהקובץ 16KB נוספים לתוך מחצית החוצץ, שתוכנה כבר נשלח ל-DSP.

נגדיר תחילה שני מבנים שיעזרו בהשמעה:

```
volatile struct {                 // WAV מבנה לשמירת סטטוס השמעת
   FILE *WAVFile ;                // WAV מצביע לקובץ
   volatile int Playing ;         // ?האם הקובץ בהשמעה עכשיו
   volatile long ToBeRead ;       // מספר הדגימות שנשאר לקרוא מהקובץ
   volatile long ToBePlayed ;     // מספר הדגימות שנשאר להשמיע
   } WAVStatus ;
```

```
struct {                  // מבנה לשמירת נתוני הזיכרון הדרושים
   short Page ;           // דף זיכרון
   short Offset;          // היסט
   byte *DMABuffer;       // מצביע לחוצץ DMA
   short RBuffer;         // מיקום הקריאה בחוצץ (מחצית ראשונה או שנייה)
   } Mem;
```

הערה

את המשתנים שבמבנה הראשון הגדרתי כ-**volatile**. volatile היא מילה שמורה, שכאשר כותבים אותה לפני משתנה, היא מחייבת את המהדר לשמור ולעדכן את ערכו תמיד בתא זיכרון קבוע ולא באוגר זמני. כיון שעלול להיווצר מצב, שגם התוכנית וגם הפסיקה ינסו לשנות בו-זמנית את ערכם של משתנים אלה, צריך יהיה להגדירם כ-volatile.

עדיין לא התחלנו לקרוא את נתוני הצלילים מהקובץ, לכן מספר הדגימות שנשאר לקרוא מהקובץ ומספר הדגימות שנשארו להשמיע, זהים לגודל בלוק הנתונים.

```
WAVStatus.ToBeRead   = BeginingOfDataBlock.BlockSize;
WAVStatus.ToBePlayed = BeginingOfDataBlock.BlockSize;
```

לפני שנתחיל לקרוא דגימות מהקובץ אל חוצץ ה-DMA, עלינו להקצות לו תחילה מקום בזיכרון. פונקציית ההקצאה צריכה להיראות כך:

```
void AssignBuffer (void)
{
   long  LinearAddress;        // כתובת ליניארית
   byte *TempBuff;             // מצביע לחוצץ זמני
   short Page1, Page2;         // שני דפי זיכרון בגודל 32KB
   ///// הקצאת 32KB של זיכרון
   TempBuff = (char *)malloc (32768);

   ///// חישוב הכתובת הליניארית
   LinearAddress = FP_SEG (TempBuff);
   LinearAddress = (LinearAddress << 4) + FP_OFF (TempBuff);

   ///// חישוב דף ראשון בתחילת החוצץ
   Page1 = LinearAddress >> 16;

   ///// חישוב דף ראשון בסוף החוצץ
   Page2 = (LinearAddress + 32767) >> 16;
```

```
    ///// בדיקה אם נחצו גבולות הדף
    if (Page1 != Page2)
    {
        Mem.DMABuffer = (char *)malloc(32768); //אם כן, הקצה חוצץ אחר
        free (TempBuff);
    }
    else
        Mem.DMABuffer = TempBuff; // אחרת, השתמש בחוצץ שהקצנו

    ///// המרת המצביע לכתובת ליניארית וקביעת הדף וההיסט /////
    LinearAddress = FP  SEG (Mem.DMABuffer);
    LinearAddress = (LinearAddress << 4) + FP_OFF (Mem.DMABuffer);
    Mem.Page = LinearAddress >> 16;
    Mem.Offset = LinearAddress & 0xFFFF;
}
```

עכשיו, כשחוצץ הזיכרון מוכן, ניתן לקרוא אליו את הדגימות מהקובץ.

הפונקציה לטעינת 16KB (חצי מגודל חוצץ DMA) מהקובץ אל החוצץ, נראית כך:

```
void ReadBuffer (short Buffer)
                  // buffer = 0 => קרא לתוך המחצית הראשונה של החוצץ
                  // buffer = 1 => קרא לתוך המחצית השנייה של החוצץ
{
    ///// אם הסתיימה הקריאה, צא מהפונקציה /////
    if (WAVStatus.ToBeRead <= 0) return;

    ///// אם החלק שיש לקרוא קטן מ-16KB , טען אותו ומלא את השאר ב"דממה" /////
    if (WAVStatus.ToBeRead < 16384)
    {
        memset(Mem.DMABuffer+(Buffer<<14),128,16384);
        fread(Mem.DMABuffer+(Buffer<<14),1,WAVStatus.ToBeRead,
              WAVStatus.WAVFile);
        WAVStatus.ToBeRead = 0;
    }
    else             // 16KB אחרת, טען אותו והחסר  מהמונה
    {
    fread (Mem.DMABuffer + (Buffer << 14), 1, 16384,
           WAVStatus.WAVFile);
    WAVStatus.ToBeRead -= 16384;
    }
    /* 16K = 14 את ערך המשתנה אנו מזיזים שמאלה 14 פעמים, כי 2 בחזקת */
}
```

לפני שמפעילים את ההשמעה האוטומטית, עלינו למלא את חוצץ ה-DMA (שגודלו 32KB). נעשה זאת על ידי טעינת 16KB למחצית הראשונה של החוצץ, ו-16KB נוספים למחצית השנייה שלו.

```
ReadBuffer (0);
ReadBuffer (1);
```

עכשיו החוצץ מוכן, ואפשר להפעיל את מעבד האותות בעזרת הפקודה d1H:

```
WriteDSP (0xD1);                          // DSP-אפשור ה
WAVStatus.Playing = 1;                    // מצב השמעה - מנגן
```

עכשיו כשה-DSP מוכן, אפשר לאתחל את ההשמעה האוטומטית:

```
void InitAutoPlayback (void)
{
```

כדי לאתחל את ההשמעה המחזורית האוטומטית, עלינו לבצע את הפעולות המפורטות להלן.

1. הכינו את ערוץ DMA לקריאה:

```
   ////// הכן את ערוץ DMA לקריאה עבור אתחול השמעה אוטומטית
   outportb (0x0A , 4|SoundBlaster.DMA);
   outportb (0x0C , 0);
   outportb (0x0B , 0x58|SoundBlaster.DMA);
```

2. שלחו את ההיסט אל בקר DMA:

```
   ///// שלח את ההיסט לבקר DMA
   outportb (SoundBlaster.DMA << 1 , Mem.Offset & 0xFF);
   outportb (SoundBlaster.DMA << 1 , Mem.Offset >> 8);
```

3. כתבו את דף הזיכרון לבקר DMA על פי הערוץ המתאים:

```
   ///// כתוב את הדף לבקר DMA
   switch (SoundBlaster.DMA)
   {
      case 0 : outportb (0x87, Mem.Page);  break;
      case 1 : outportb (0x83, Mem.Page);  break;
      case 3 : outportb (0x82, Mem.Page);  break;
   }
```

4. קבעו את אורך חוצץ ה-DMA ל- 32KB :

```
   ///// (32KB) 7fffH-קבע את אורך בלוק ה-DMA ל
   outportb ((SoundBlaster.DMA << 1) + 1, 0xFF);
   outportb ((SoundBlaster.DMA << 1) + 1, 0x7F);
   outportb (0x0A, SoundBlaster.DMA);
```

5. קבעו את אורך חוצץ ה-DSP ל- 16KB :

```
   ///// (16KB) 3fffH-קבע את אורך הבלוק של DSP ל
   WriteDSP (0x48);
   WriteDSP (0xFF);
   WriteDSP (0x3F);
```

6. הפעילו את ההשמעה בעזרת הפקודה 1cH של DSP :

```
   ///// (auto playback) התחל את ההשמעה האוטומטית
   WriteDSP (0x1C);
}
```

לאחר שמאתחלים את ההשמעה המחזורית האוטומטית, מתחיל ה-DSP להשמיע את 16KB הראשונים. ברגע שהוא מסיים, הוא מפעיל את בקשת הפסיקה (IRQ), המכינה עבורו את ה-16KB הבאים.

כיצד התהליך מסתיים?

נכתוב תחילה פונקציה לעצירת ההשמעה המחזורית. הפונקציה צריכה לבצע שלוש פעולות :

1. לעצור את השידור מערוץ DMA (נעשה זאת על ידי הפקודה d0H של DSP).
2. להכניס "שקר" למשתנה המציג את סטטוס ההשמעה.
3. לסגור את קובץ WAV.

הפונקציה תיראה כך :

```
void StopPlayback (void)
{
   WriteDSP (0xD0);                    // DMA עצור את השידור מערוץ
   WAVStatus.Playing = 0;              // לא נותר עוד מה להשמיע
   fclose (WAVStatus.WAVFile);         // WAV סגור את קובץ
}
```

עכשיו אנו יודעים כיצד להפעיל את ההשמעה המחזורית וכיצד לעצור אותה, אך כיצד נדע מתי לעצור אותה? כדי שההשמעה תיעצר לבד (ללא התערבות התוכנית), היא צריכה לעקוב בעצמה אחר ההתקדמות. כיצד אפשר לעשות זאת?

התשובה פשוטה למדי. אנו יודעים שלאחר כל השמעה של 16KB, ה-DSP מפעיל את בקשת הפסיקה. אם כך, נוכל לכתוב בעצמנו תוכנית שירות לפסיקה שגם תעדכן את מונה הדגימות שנותר להשמיע (ToBePlayed), ואם לא נותר עוד מה להשמיע, היא תעצור את ההשמעה המחזורית. את תוכנית הפסיקה שנכתוב, נציב במקום הפסיקה המקורית, והיא תופעל על ידי מעבד האותות.

תוכנית השירות צריכה להיראות כך:

```
void interrupt ServiceIRQ (void)
{
   ///// DSP-שחרר את פורט 8 הסיביות של ה
   inportb (SoundBlaster.Base + 0xE);

   ///// הכירו בפסיקת החומרה
   outportb (0x20, 0x20);
```

בקווי בקשת הפסיקה (IRQ) 2, 10 ו-11, הפסיקה גולשת ולכן יש לכתוב:

```
   ///// (11-10,2 הכר בפסיקה הגולשת (עבור קווי בקשת פסיקה
   if (SoundBlaster.IRQ==2 || SoundBlaster.IRQ==10 ||
                             SoundBlaster.IRQ==11)
      outportb (0xA0, 0x20);
```

תוכנית השירות של הפסיקה צריכה לטפל בחוצצים ובמבנה הסטטוס.

תחילה נפחית 16KB ממונה הדגימות שנשארו להשמעה (ToBePlayed), כי הפסיקה מופעלת לאחר שה-DSP מסיים להשמיע 16KB. אם נשאר עוד מה להשמיע (אם לאחר הפחתה של 16KB, ערך מונה הדגימות שנשארו להשמעה עדיין חיובי), עלינו לטעון לחוצץ ה-DMA את ה-16KB. אם זהו המחזור האחרון שיש לקרוא (ערך מונה הדגימות שנותר להשמיע קטן או שווה ל-16KB), נשלח ל-DSP את הפקודה daH המודיעה שצריך להפעיל את תוכנית הפסיקה פעם אחת נוספת בלבד. אחרת (אם אין עוד מה להשמיע), נפעיל את הפונקציה לעצירת ההשמעה האוטומטית:

```
   ///// טיפול בחוצצים
   if (WAVStatus.Playing)               // אם מנגן
   {
      WAVStatus.ToBePlayed -= 16384;    // הורד 16KB מהמונה
      if (WAVStatus.ToBePlayed > 0)     // אם נשאר מה לנגן
      {
         ReadBuffer (Mem.RBuffer);      // קרא מחזור לחוצץ
         if (WAVStatus.ToBePlayed <= 16384)// אם זהו המחזור האחרון
            WriteDSP (0xDA);
      }
      else
         StopPlayback();
   }
```

בסיום הפסיקה, עלינו לדאוג ש-Rbuffer יצביע על מחצית החוצץ הבאה.

שימו לב! אם הוא מצביע על המחצית הראשונה – ערכו 0, ויש להפוך אותו ל-1 כדי שיצביע על המחצית השנייה. אם Rbuffer מצביע על המחצית השנייה – ערכו 1, ויש להפוך אותו לאפס כדי שיצביע על המחצית הראשונה. נעשה זאת כך:

```
  Mem.RBuffer ^= 1;
}              // IRQ-סוף תוכנית השירות לפסיקה – ה
```

עכשיו יש לנו תוכנית מוכנה לטיפול בפסיקה. עלינו לשמור אותה בתא המתאים בווקטור הפסיקות, כדי שמעבד האותות יוכל להפעיל אותה. מיקום התא נקבע כך:

- אם קו בקשת הפסיקה (IRQ) הוא 2, מעבד האותות DSP יפעיל את הפסיקה שבתא 71H.
- אם קו IRQ הוא 10, מעבד האותות יפעיל את הפסיקה שבתא 72H.
- אם קו IRQ הוא 11, מעבד האותות יפעיל את הפסיקה שבתא 73H.
- עבור כל שאר קווי בקשות הפסיקה, מעבד האותות יפעיל את הפסיקה שבתא 8 + מספר קו IRQ.

תזכורת: לפני שמירת פסיקה בתא מסוים בווקטור הפסיקות, תמיד חשוב לשמור את תוכן התא הנוכחי בתא ריק, כדי שניתן יהיה לשחזר אותו בתום תוכנית הפסיקה.

במקרה שלנו, אם לדוגמה תוכנה אחרת משמיעה קובץ WAV, והתוכנית שלנו, המופעלת אחריה, תרצה להשמיע קובץ WAV אחר (ולא נשמור את התוכן המקורי של התא בתא ריק אחר), השמעת קובץ WAV של התוכנה החיצונית תיעצר ולא נוכל לשחזר את הפעולה. אך אם נשמור את התוכן המקורי של התא בתא ריק ונשחזר אותו בתום השמעת קובץ WAV שלנו, השמעת קובץ WAV של התוכנה החיצונית תתחדש בסיום השמעת הקובץ שלנו, כאילו דבר לא קרה.

אם כך, פונקציית אתחול תוכנית השירות לפסיקה צריכה לבצע שלוש פעולות:

1. לשמור את הפסיקה הנוכחית בתא ריק (לדוגמה 104), "תא ביניים".
2. לקבוע את הפסיקה שכתבנו בתא המתאים.
3. לאפשר את ה-IRQ (Enable IRQ).

הפונקציה תיכתב כך:

```
void InitIRQ (void interrupt (*NewISR)())
{
   ///// 104 שמור את הפיסקה הנוכחית בתא ריק מספר
   switch (SoundBlaster.IRQ)
   {
      case  2: setvect(104,getvect (0x71));  break;
      case 10: setvect(104,getvect (0x72));  break;
```

```
      case 11: setvect(104,getvect (0x73));   break;
      default: setvect(104,getvect (8+SoundBlaster.IRQ));
   }

   ///// קבע את הפסיקה שלנו כפסיקת ה-IRQ
   switch (SoundBlaster.IRQ)
   {
      case  2: setvect (0x71, NewISR);  break;
      case 10: setvect (0x72, NewISR);  break;
      case 11: setvect (0x73, NewISR);  break;
      default: setvect (8+SoundBlaster.IRQ, NewISR);
   }

   ///// אפשר את ה-IRQ
   switch (SoundBlaster.IRQ)
   {
      case  2: outportb (0xA1, inportb(0xA1)&253);  break;
      case 10: outportb (0xA1, inportb(0xA1)&251);  break;
      case 11: outportb (0xA1, inportb(0xA1)&247);  break;
      default: outportb (0x21, inportb(0x21)& !(1 <<
                                      SoundBlaster.IRQ));
   }
   if (SoundBlaster.IRQ==2 || SoundBlaster.IRQ==10 ||
                                     SoundBlaster.IRQ==11)
      outportb (0x21, inportb(0x21) & 251);
}
```

הפונקציה לשחזור הפסיקה המקורית צריכה לבצע פעולות אלו:

1. להעתיק את הפסיקה מתוך תא הביניים חזרה אל התא המקורי.
2. לאפשר את ה-IRQ (Enable IRQ).

הפונקציה תיכתב כך:

```
void RestoreIRQ (void)
{
   /// שחזר את הפסיקה המקורית ששמרת בתא 104
   switch (SoundBlaster.IRQ)
   {
      case  2: setvect (0x71, getvect(104));   break;
      case 10: setvect (0x72, getvect(104));   break;
      case 11: setvect (0x73, getvect(104));   break;
      default: setvect (8+SoundBlaster.IRQ, getvect(104));
   }
```

```
   ///// IRQ-אפשר את ה
   switch (SoundBlaster.IRQ)
   {
      case  2: outportb (0xA1, inportb(0xA1) | 2);  break;
      case 10: outportb (0xA1, inportb(0xA1) | 4);  break;
      case 11: outportb (0xA1, inportb(0xA1) | 8);  break;
      default: outportb (0x21, inportb(0x21) |
                                   (1<<SoundBlaster.IRQ));
   }
   if (SoundBlaster.IRQ==2 || SoundBlaster.IRQ== 10 ||
                                      SoundBlaster.IRQ==11)
      outportb (0x21, inportb(0x21) | 4);
}
```

בכך סיימנו להרכיב את מנגנון ההשמעה המחזורית. מהרגע שנפעיל אותו בעזרת פונקציית אתחול ההשמעה המחזורית, הוא יידע לפעול בעצמו ולעצור ברגע הנכון ללא תלות בתוכנית. ממש "פרפטום מובילה" (חוזר עד אין סוף).

15.5 השמעת מחזור בודד

יש מקרה אחד שבו המנגנון שכתבנו בסעיף הקודם לא יפעל כראוי, נסו לחשוב מהו!

אם גודל בלוק נתוני ההשמעה בקובץ WAV אינו גדול מ-16KB (מחזור אחד), תוכנית הפסיקה שלנו לא תפעל בסופו.

מכיון שאנו פועלים עם קבצי WAV מסוג 8 סיביות מונו (החסכוניים מאוד בנפחם), מצב זה ייתכן בהחלט. לשם כך נכתוב פונקציה להשמעת מחזור בודד. אם גודל בלוק נתוני ההשמעה בקובץ WAV אינו גדול מ-16KB, נפעיל פונקציה זו, ולא את פונקציית אתחול ההשמעה האוטומטית.

הפונקציה צריכה לבצע את הפעולות הבאות:

1. להפעיל את פונקציית ההשמעה המחזורית (למרות שזו אינה השמעה מחזורית, אנו זקוקים בכל זאת לרוב הפקודות שפיתחנו עבור הפונקציה לאתחול ההשמעה המחזורית, אשר מטפלות ב-DMA).
2. להגדיר את אורך מחזור ההשמעה כאורך הבלוק שברצוננו להשמיע (בפונקציית אתחול ההשמעה המחזורית השתמשנו בפקודה 48H של DSP, כדי לקבוע שאורך כל מחזור יהיה 16KB. הפעם נשתמש בפקודה באותו אופן, כדי להגדיר את אורך המחזור הבודד כאורך הבלוק שברצוננו להשמיע).
3. להפעיל את השמעת המחזור הבודד. נעשה זאת בעזרת הפקודה 1cH של DSP.
4. נאפס את מונה הדגימות שנותרו להשמעה (ToBePlayed).

הפונקציה תיראה כך :

```
void SingleCyclePlayback (void)
{
   InitAutoPlayback();
   WriteDSP (0x48);
   WriteDSP (WAVStatus.ToBePlayed & 0xFF);
   WriteDSP (WAVStatus.ToBePlayed >> 8);

   WriteDSP (0x1c);              // הפעלת מחזור השמעה בודד

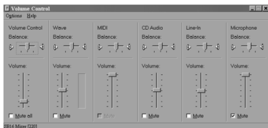
   WAVStatus.ToBePlayed = 0;    // אין עוד מה להשמיע

}
```

15.6 עוצמת הקול

הביטו בתרשים 15.3 המציג את מד כיוון העוצמה של מערכת ההפעלה Windows. שימו לב שניתן לכוון בנפרד את העוצמה ואת איזון הרמקולים בכל רכיב, וגם את העוצמה ואת האיזון הכללי MASTER.

כך נוכל לעשות בקלות גם בתוכנית שלנו.



תרשים 15.3

כדי לקבוע את העוצמה והאיזון של אחד הרכיבים, עלינו לשלוח ליציאה 224H את קוד הרכיב, ומייד אחר כך לשלוח ליציאה 225H את משתנה שקלול העוצמה.

הרכיבים המעניינים אותנו כאן:

1. עוצמת השמעת קבצי WAV – קוד 4H.
2. עוצמת המאסטר – קוד 22H.

נגדיר את הקבועים האלה עבור התוכנית שלנו:

```
#define MASTERV_VOL 0x22
#define WAV_VOL 0x04
```

משתנה שקלוט העוצמה הוא בגודל בית. ארבע הסיביות הראשונות בו (מימין) מייצגות את עוצמת הקול ברמקול הימני, וארבע הסיביות האחרונות מייצגות את עוצמת הקול ברמקול השמאלי.

הפונקציה לקביעת עוצמת הקול נראית כך:

```
void VolumeSet(byte Where , byte Vol)
{
  // Where - באיזה רכיב , Vol - משתנה שקלוט העוצמה
  outportb(0x224,Where);
  outportb(0x225,Vol);
}
```

כדי לברר מהי עוצמת הקול הנוכחית, עלינו לשלוח ליציאה 224H את קוד הרכיב, ומייד אחר כך לקרוא מיציאה 225H את משתנה שקלוט העוצמה.

הפונקציה תיראה כך:

```
byte VolumeGet(byte Where)
{
   outportb(0x224,Where);
   return ( inportb(0x225) );
}
```



נסו ליצור בקר לשליטה על עוצמת הקול ולאיזון הרמקולים, אשר ייראה כמו הבקר של Windows (תרשים 15.3).

רמז: היעזרו בתוכנית הדוגמה של נגן התקליטורים.

זו כל התורה להשמעת קבצי WAV. אני מניח שעכשיו אתם די מבולבלים מכל מה שקראתם עד כה ומרגישים כאילו אתם "לא יודעים מה נפל עליכם". אז אל דאגה, קחו נשימה עמוקה וקראו בעיון את יחידת התכנות המפורטת שבסעיף הבא. אני מבטיח שהכל יהיה הרבה יותר ברור אחרי שתראו כיצד הכל מתחבר.

15.7 יחידת הספריה להשמעת קבצי WAV

ממשק היחידה (Unit interface)

```
/*****************************************************************************
 *                                W A V . H                                  *
 *                            ~~~~~~~~~~~~~~~~~~                             *
 *                  ממשק ליחידה להשמעת קבצי מוסיקה מסוג WAV                   *
 *                                                                           *
 *  הערות : 1) היחידה תזהה אוטומטית את כרטיס הקול ותמצא את כתובתו,            *
 *             את ה-IRQ ואת ערוץ DMA.                                         *
 *          2) היחידה תומכת בקבצי WAV בפורמט 8 סיביות Mono סטנדרטי.            *
 *          3) הטיפול בהשמעת קבצי הקול מתבצע באמצעות פסיקות, לכן ההשמעה       *
 *             לא תפריע למהלך התוכנית !                                       *
 ****************************************************************************/

#define MASTERV_VOL 0x22
#define WAV_VOL 0x04

typedef unsigned char  byte;                  // משתנה חיובי בגודל בית

int CheckBaseAddress (short AddressToCheck);
/*---------------------------------------------------------------------------
        פונקציה לבדיקת כתובת בסיס.
        פרמטרים מועברים :
           AddressToCheck - הכתובת לבדיקה.
        ערך מוחזר :
           אמת אם הכתובת שנבדקה היא אכן כתובת הבסיס של ה-DSP.
 ---------------------------------------------------------------------------*/

void WriteDSP(byte Value);
/*---------------------------------------------------------------------------
        פונקציה לשליחת ערך כלשהו ל-DSP (Digital Signal Processor).
        פרמטרים מועברים :
           Value - הערך שאותו אנו רוצים לכתוב ל- DSP.
        ערך מוחזר - אין.
 ---------------------------------------------------------------------------*/
```

```
int FindSoundBlaster(void);
/*---------------------------------------------------------------------------
                              פונקציה למציאת כרטיס הקול, ה-IRQ, וה-DMA.
                                                   פרמטרים מועברים - אין.
                                      ערך מוחזר - אמת אם הזיהוי התבצע בהצלחה.
 ---------------------------------------------------------------------------*/

void VolumeSet(byte Where , byte Vol);
/*---------------------------------------------------------------------------
                                                       פונקציה לקביעת עוצמת קול.
                                                            פרמטרים מועברים :
                               Where - הרכיב שאת עוצמת קולו אנו רוצים לשנות.
                                          Vol - בית המציין את עוצמת הקול.
    (ארבע הסיביות הראשונות מייצגות את הרמקול הימני, והאחרונות את השמאלי)
                                                          ערך מוחזר : אין.
 ---------------------------------------------------------------------------*/

byte VolumeGet(byte Where);
/*---------------------------------------------------------------------------
                                                       פונקציה לבדיקת עוצמת קול.
                                                            פרמטרים מועברים :
                                 Where - הרכיב שאת עוצמת קולו אנו רוצים לדעת.
                                    ערך מוחזר - בית המציין את עוצמת הקול.
  (ארבע הסיביות הראשונות מייצגות את הרמקול הימני, והאחרונות - את השמאלי)
 ---------------------------------------------------------------------------*/

void InitAutoPlayback (void);
/*---------------------------------------------------------------------------
                                     פונקציה לאתחול השמעה מחזורית אוטומטית.
                                                   פרמטרים מועברים - אין.
                                                        ערך מוחזר -  אין.
 ---------------------------------------------------------------------------*/

void SingleCyclePlayback (void);
/*---------------------------------------------------------------------------
                                פונקציה לטיפול בהשמעת מחזור בודד (יחיד).
                                                   פרמטרים מועברים - אין.
                                                        ערך מוחזר -  אין.
 ---------------------------------------------------------------------------*/
```

```
void AssignBuffer (void);
/*------------------------------------------------------------------------
    פונקציה להקצאת חוצץ זיכרון עבור הנתונים להשמעה.
    פרמטרים מועברים - אין.
    ערך מוחזר -  אין.
 ------------------------------------------------------------------------*/

void ReadBuffer (short Buffer);
/*------------------------------------------------------------------------
    פונקציה לטעינת16KB (חצי מחוצץ DMA) מהקובץ אל החוצץ.
    פרמטרים מועברים :
     Buffer - איזו מחצית חוצץ עלינו לקרוא.
     ערך מוחזר -  אין.
 ------------------------------------------------------------------------*/

void StopPlayback (void);
/*------------------------------------------------------------------------
    פונקציה לעצירת ההשמעה.
    פרמטרים מועברים - אין.
    ערך מוחזר -  אין.
 ------------------------------------------------------------------------*/

void interrupt ServiceIRQ (void);
/*------------------------------------------------------------------------
    פסיקה המטפלת בהשמעה המחזורית, הפסיקה תופעל אוטומטית בסוף כל מחזור.
    פרמטרים מועברים - אין.
    ערך מוחזר -  אין.
 ------------------------------------------------------------------------*/

void InitIRQ (void interrupt (*NewISR)());
/*------------------------------------------------------------------------
    פונקציה לאתחול פסיקת ההשמעה המחזורית.
    פרמטרים מועברים - הפסיקה שלנו.
    ערך מוחזר -  אין.
 ------------------------------------------------------------------------*/

void RestoreIRQ (void);
/*------------------------------------------------------------------------
    פונקציה לשחזור פסיקת ההשמעה המחזורית המקורית.
    פרמטרים מועברים - אין.
    ערך מוחזר       - אין.
 ------------------------------------------------------------------------*/
```

```
int InitSoundBlaster (void);
/*---------------------------------------------------------------------------
                                            פונקציה לאתחול כרטיס הקול.
                                                פרמטרים מועברים - אין.
                              ערך מוחזר -  אמת אם האתחול הסתיים בהצלחה.
 ---------------------------------------------------------------------------*/

void CloseSoundBlaster (void);
/*---------------------------------------------------------------------------
  פונקציה ליציאה מהיחידה (שחרור הזיכרון שהוקצה ושחזור הפסיקה המקורית).
                                                פרמטרים מועברים - אין.
                                                   ערך מוחזר -  אין.
 ---------------------------------------------------------------------------*/

int PlayWAV (char *FileName);
/*---------------------------------------------------------------------------
                                              פונקציה להשמעת קובץ WAV.
                                                     פרמטרים מועברים :
                                        FileName - שם קובץ ה-WAV.
                                                          ערך מוחזר :
                                      1  > ההשמעה הופעלה בהצלחה.
                      -1 > אין מספיק זיכרון, או שכרטיס הקול לא אותחל.
                                         -2 > קובץ WAV אינו קיים.
                                  -3 > הקובץ אינו קובץ WAV תקין.
 ---------------------------------------------------------------------------*/

int WAVPlaying ();
/*---------------------------------------------------------------------------
                              פונקציה הבודקת האם מתנגן כרגע קובץ כלשהו.
                                                פרמטרים מועברים - אין.
                                                ערך מוחזר - אמת אם כן.
 ---------------------------------------------------------------------------*/
```

מימוש היחידה

```
/*************************************************************************
 *                               W A V . C                               *
 *                          ~~~~~~~~~~~~~~~~~~                           *
 *               יחידה להשמעת קבצי מוסיקה מסוג WAV                        *
 *                                                                       *
 * הערות : 1) היחידה תבצע זיהוי אוטומטי לכרטיס הקול ותמצא את כתובתו,      *
 *            את ה-IRQ ואת ערוץ DMA.                                     *
 *         2) היחידה תומכת בקבצי WAV בפורמט 8 סיביות Mono סטנדרטי.        *
 *         3) הטיפול בהשמעת קבצי הקול מתבצע באמצעות פסיקות, לכן ההשמעה    *
 *            לא תפריע למהלך התוכנית !                                   *
 *************************************************************************/

//////////////////////////////////////////////////////////////////////////
//                          ה צ ה ר ו ת                                 //
//////////////////////////////////////////////////////////////////////////

#include "dos.h"
#include "stdio.h"
#include "stdlib.h"

#define MASTERV   VOL 0x22
#define WAV   VOL 0x04

typedef unsigned char  byte;                  // משתנה חיובי בגודל בית
struct {                     // בלוק RIFF
   long BlockType;           // קוד הזיהוי של הבלוק (צ"ל - 0x46464952)
   long BlockSize;           // גודל הבלוק
   long RIFF;                // ערך RIFF (בקובץ WAV תקין צ"ל - 0x45564157)
  } RIFFBlock;
struct {                     // בלוק של פורמט קובץ WAV
   long BlockType;           // קוד הזיהוי של הבלוק (צ"ל - 0x20746D66)
   long BlockSize;           // גודל הבלוק
   short WaveType;           // סוג קובץ WAV
   short Channels;           // מספר ערוצי השמע
   long SampleRate;          // קצב הדגימה
   long BytesPerSecond;      // בתים לשנייה (BPS)
   short BlockAlignment;     // יישור הסיביות בבלוק הנתונים
   short BitResolution;      // רזולוציית הסיביות
  } WAVFormatBlock;
```

```
struct {                    // הנתונים להשמעה של תחילת בלוק
   long BlockType;          // קוד הזיהוי של הבלוק (צ"ל - 0x61746164)
   long BlockSize;          // גודל הבלוק
  } BeginingOfDataBlock;
struct {                    // מבנה לשמירת הנתונים על כרטיס הקול
   char DMA ;               // ערוץ הגישה הישירה לזיכרון
   char IRQ ;               // רמת ערוץ בקשת הפסיקה
   short Base ;             // כתובת הבסיס של ה-Sound Blaster
   } SoundBlaster ;
volatile struct {             // מבנה לשמירת סטטוס השמעת WAV
   FILE *WAVFile ;            // מצביע לקובץ WAV
   volatile int Playing ;     // האם הקובץ בהשמעה עכשיו?
   volatile long ToBeRead ;   // מספר הדגימות שנשאר לקרוא מהקובץ
   volatile long ToBePlayed ; // מספר הדגימות שנשאר להשמיע
   } WAVStatus ;
struct {                      // מבנה לשמירת נתוני הזיכרון הדרושים
   short Page ;               // דף זיכרון
   short Offset;              // היסט
   byte *DMABuffer;           // מצביע לחוצץ DMA
   short RBuffer;             // מיקום הקריאה בחוצץ
   } Mem;

////////////////////////////////////////////////////////////////////////////////
//                                פ ו נ ק צ י ו ת                             //
////////////////////////////////////////////////////////////////////////////////

int CheckBaseAddress (short AddressToCheck)
/*---------------------------------------------------------------------------
                                                     פונקציה לבדיקת כתובת בסיס.
                                                             פרמטרים מועברים :
                                           AddressToCheck - הכתובת לבדיקה.
                                                                  ערך מוחזר :
                         אמת אם הכתובת שנבדקה היא אכן כתובת הבסיס של ה-DSP.
 ---------------------------------------------------------------------------*/
{
   outportb (AddressToCheck+0x6,1);
   delay (10);
   outportb (AddressToCheck+0x6,0);
   delay (10);
```

```
if (  ( ( inportb(AddressToCheck+0xE)&0x80 ) == 0x80 ) &&
        ( inportb(AddressToCheck+0xA) == 0xAA )  )
   {
      אם הכתובת שנבדקה היא אכן כתובת הבסיס של ה-DSP /////
      SoundBlaster.Base = AddressToCheck ;   // עדכן את מבנה כרטיס הקול
      return (1) ;                           // והחזר 'אמת'
   }
                                             /// אחרת, החזר 'שקר'
   return 0;
}

void WriteDSP(byte Value)
/*------------------------------------------------------------------------
            פונקציה לשליחת ערך כלשהו ל-DSP (Digital Signal Processor).
            פרמטרים מועברים :
               Value - הערך שאותו אנו רוצים לכתוב ל- DSP.
            ערך מוחזר -  אין.
 ------------------------------------------------------------------------*/
{
   המתן שה-DSP יהיה זמין /////
   while ((inportb (SoundBlaster.Base + 0xC) & 0x80) == 0x80);
   שליחת הערך ל-DSP /////
   outportb (SoundBlaster.Base + 0xC, Value);
}

int FindSoundBlaster(void)
/*------------------------------------------------------------------------
            פונקציה למציאת כרטיס הקול, ה-IRQ, וה-DMA.
            פרמטרים מועברים - אין.
            ערך מוחזר - אמת אם הזיהוי התבצע בהצלחה.
 ------------------------------------------------------------------------*/
{
   char i=1;
   char *BLASTER;
   int Found = 0 ;
   SoundBlaster.Base = 0;            // עדיין לא נמצאה כתובת כלשהי
   /////  חיפוש כתובת של ה- SoundBlaster
   while (i<9 && !Found)
   {
      if (CheckBaseAddress (0x200 + (i << 4)))
         Found = 1;
      i++;
   }
```

```
   if (i == 9)
      return(0);                      // אם לא נמצא, החזר שקר
   ///// חיפוש IRQ ו-DMA במחרוזת ה-environment של ה-SoundBlaster
   BLASTER = getenv ("BLASTER");
   SoundBlaster.DMA = 0;
   for (i = 0; i < strlen (BLASTER); i++)
   if ((BLASTER [i] | 32) == 'd')
      SoundBlaster.DMA = BLASTER [i + 1] - '0';
   for (i = 0; i < strlen (BLASTER); i++)
      if ((BLASTER [i] | 32) == 'i')
      {
         SoundBlaster.IRQ = BLASTER [i + 1] - '0';
         if (BLASTER [i + 2] != ' ')
         {
            SoundBlaster.IRQ *= 10 ;
            SoundBlaster.IRQ += BLASTER[i + 2] - '0';
         }
      }
   return (1);
}

void VolumeSet(byte Where , byte Vol)
/*--------------------------------------------------------------------------
                                                 פונקציה לקביעת עוצמת קול.
                                                          פרמטרים מועברים :
                            Where - הרכיב שאת עוצמת קולו אנו רוצים לשנות.
                                            Vol - בית המציין את עוצמת הקול.
   (ארבע הסיביות הראשונות מייצגות את הרמקול הימני, והאחרונות את השמאלי)
                                                          ערך מוחזר : אין.
 --------------------------------------------------------------------------*/
{
   outportb(0x224,Where);
   outportb(0x225,Vol);
}

byte VolumeGet(byte Where)
/*--------------------------------------------------------------------------
                                                 פונקציה לבדיקת עוצמת קול.
                                                          פרמטרים מועברים :
                             Where - הרכיב שאת עוצמת קולו אנו רוצים לדעת.
                                      ערך מוחזר - בית המציין את עוצמת הקול.
   (ארבע הסיביות הראשונות מייצגות את הרמקול הימני, והאחרונות את השמאלי)
 --------------------------------------------------------------------------*/
```

```
{
   outportb(0x224,Where);
   return ( inportb(0x225) );
}

void StopPlayback (void)
/*--------------------------------------------------------------------------
                                   פונקציה לעצירת ההשמעה.
                                   פרמטרים מועברים - אין.
                                   ערך מוחזר -  אין.
 --------------------------------------------------------------------------*/
{
   WriteDSP (0xD0);                    // עצור את השידור מערוץ ה-DMA
   WAVStatus.Playing = 0;              // לא נותר עוד מה להשמיע
   fclose (WAVStatus.WAVFile);         // סגור את קובץ ה-WAV
}

void InitAutoPlayback (void)
/*--------------------------------------------------------------------------
                                   פונקציה לאתחול השמעה מחזורית אוטומטית.
                                   פרמטרים מועברים - אין.
                                   ערך מוחזר -  אין.
 --------------------------------------------------------------------------*/
{
   ///// הכן את ערוץ ה-DMA לקריאה עבור אתחול השמעה אוטומטית
   outportb (0x0A , 4|SoundBlaster.DMA);
   outportb (0x0C , 0);
   outportb (0x0B , 0x58|SoundBlaster.DMA);
   ///// שליחת ההיסט לבקר ה-DMA
   outportb (SoundBlaster.DMA << 1 , Mem.Offset & 0xFF);
   outportb (SoundBlaster.DMA << 1 , Mem.Offset >> 8);
   ///// כתיבת הדף לבקר ה-DMA
   switch (SoundBlaster.DMA)
   {
      case 0 : outportb (0x87, Mem.Page);  break;
      case 1 : outportb (0x83, Mem.Page);  break;
      case 3 : outportb (0x82, Mem.Page);  break;
   }
   ///// קביעת אורך הבלוק של ה-DMA ל-7fffH (32KB)
   outportb ((SoundBlaster.DMA << 1) + 1, 0xFF);
   outportb ((SoundBlaster.DMA << 1) + 1, 0x7F);
   outportb (0x0A, SoundBlaster.DMA);
```

```
   קביעת אורך הבלוק של ה-DSP ל-3fffH (16KB) /////
   WriteDSP (0x48);
   WriteDSP (0xFF);
   WriteDSP (0x3F);
   התחל את ההשמעה האוטומטית (auto playback) /////
   WriteDSP (0x1C);
}

void SingleCyclePlayback (void)
/*--------------------------------------------------------------------------
      פונקציה לטיפול בהשמעת מחזור בודד (יחיד).
      פרמטרים מועברים - אין.
      ערך מוחזר -  אין.
 --------------------------------------------------------------------------*/
{
   WAVStatus.ToBePlayed--;
   InitAutoPlayback();
   WriteDSP (0x48);
   WriteDSP (WAVStatus.ToBePlayed & 0xFF);
   WriteDSP (WAVStatus.ToBePlayed >> 8);
   הפעלת מחזור השמעה בודד /////
   WriteDSP (0x1c);
   אין עוד מה להשמיע /////
   WAVStatus.ToBePlayed = 0;
}

void AssignBuffer (void)
/*--------------------------------------------------------------------------
      פונקציה להקצאת חוצץ זיכרון עבור הנתונים להשמעה.
      פרמטרים מועברים - אין.
      ערך מוחזר -  אין.
 --------------------------------------------------------------------------*/
{
   long  LinearAddress;       // כתובת ליניארית
   byte *TempBuff;            // מצביע לחוצץ זמני
   short Page1, Page2;        // שני דפי זיכרון בגודל 32KB
   הקצאת 32KB של זיכרון /////
   TempBuff = (char *)malloc (32768);
   חישוב הכתובת הליניארית /////
   LinearAddress = FP_SEG (TempBuff);
   LinearAddress = (LinearAddress << 4) + FP_OFF (TempBuff);
```

```
   ///// חישוב דף ראשון בתחילת החוצץ
   Page1 = LinearAddress >> 16;
   ///// חישוב דף ראשון בסוף החוצץ
   Page2 = (LinearAddress + 32767) >> 16;
   ///// בדיקה האם נחצו גבולות הדף
   if (Page1 != Page2)
   {
                              // אם כן, הקצה חוצץ אחר.
      Mem.DMABuffer = (char *)malloc(32768);
      free (TempBuff);
   }
   else
                              // אחרת השתמש בחוצץ שהקצינו
      Mem.DMABuffer = TempBuff;
   ///// המרת המצביע לכתובת ליניארית וקביעת הדף וההיסט
   LinearAddress = FP_SEG (Mem.DMABuffer);
   LinearAddress = (LinearAddress << 4) + FP_OFF (Mem.DMABuffer);
   Mem.Page = LinearAddress >> 16;
   Mem.Offset = LinearAddress & 0xFFFF;
}

void ReadBuffer (short Buffer)
/*-------------------------------------------------------------------------
                  פונקציה לטעינת 16KB (חצי מחוצץ DMA) מהקובץ אל החוצץ.
                                                    פרמטרים מועברים :
                          Buffer - איזו מחצית חוצץ עלינו לקרוא.
                                                    ערך מוחזר -  אין.
 -------------------------------------------------------------------------*/
{
   ///// אם הסתיימה הקריאה צא מהפונקציה
   if (WAVStatus.ToBeRead <= 0) return;
   ///// אם החלק שיש לקרוא קטן מ-16KB טען אותו, ומלא את השאר ב"דממה",
   if (WAVStatus.ToBeRead < 16384)
   {
      memset(Mem.DMABuffer+(Buffer<<14),128,16384);
      fread(Mem.DMABuffer+(Buffer<<14),1,WAVStatus.ToBeRead,
         WAVStatus.WAVFile);
      WAVStatus.ToBeRead = 0;
   }
```

```
   else                         // אחרת, טען אותו והחסר 16KB מהמונה
   {
   fread (Mem.DMABuffer + (Buffer << 14), 1, 16384, WAVStatus.WAVFile);
   WAVStatus.ToBeRead -= 16384;
   }

   /*  16K = 14 את ערך המשתנה אנו מסובבים שמאלה 14 פעמים כי 2 בחזקת */
}

void interrupt ServiceIRQ (void)
/*----------------------------------------------------------------------
      פסיקה המטפלת בהשמעה המחזורית, הפסיקה תופעל אוטומטית בסוף כל מחזור.
                                              פרמטרים מועברים - אין.
                                                    ערך מוחזר -  אין.
 ----------------------------------------------------------------------*/
{
   ///// DSP שחרר את פורט 8 הסיביות של
   inportb (SoundBlaster.Base + 0xE);
   ///// הכר בפסיקת החומרה
   outportb (0x20, 0x20);
   ///// (11-10,2 ו IRQ הכר בפסיקה הגולשת (עבור
   if (SoundBlaster.IRQ==2||SoundBlaster.IRQ==10||SoundBlaster.IRQ==11)
      outportb (0xA0, 0x20);
   ///// טיפול בחוצצים
   if (WAVStatus.Playing)                       // אם מנגן
   {
      WAVStatus.ToBePlayed -= 16384;            // הורד 16KB מהמונה
      if (WAVStatus.ToBePlayed > 0)             // אם נשאר מה לנגן
      {
         ReadBuffer (Mem.RBuffer);              // קרא מחזור לחוצץ
      if (WAVStatus.ToBePlayed <= 16384)        // אם זהו המחזור האחרון
            WriteDSP (0xDA);
      }
      else
         StopPlayback();
   }
  Mem.RBuffer ^= 1;
}
```

```
void InitIRQ (void interrupt (*NewISR)())
/*------------------------------------------------------------------------
                                          פונקציה לאתחול פסיקת ההשמעה המחזורית.
                                                   פרמטרים מועברים - הפסיקה שלנו.
                                                                 ערך מוחזר -  אין.
 ------------------------------------------------------------------------*/
{
   ///// שמירת הפיסקה הנוכחית בתא ריק מספר 104
   switch (SoundBlaster.IRQ)
   {
      case  2: setvect(104,getvect (0x71));   break;
      case 10: setvect(104,getvect (0x72));   break;
      case 11: setvect(104,getvect (0x73));   break;
      default: setvect(104,getvect (8+SoundBlaster.IRQ));
   }
   ///// קביעת הפסיקה שלנו כפסיקת ה-IRQ
   switch (SoundBlaster.IRQ)
   {
      case  2: setvect (0x71, NewISR);  break;
      case 10: setvect (0x72, NewISR);  break;
      case 11: setvect (0x73, NewISR);  break;
      default: setvect (8+SoundBlaster.IRQ, NewISR);
   }
   ///// אפשר את ה-IRQ
   switch (SoundBlaster.IRQ)
   {
      case  2: outportb (0xA1, inportb(0xA1)&253);  break;
      case 10: outportb (0xA1, inportb(0xA1)&251);  break;
      case 11: outportb (0xA1, inportb(0xA1)&247);  break;
      default: outportb (0x21, inportb(0x21)& !(1<<SoundBlaster.IRQ));
   }
   if (SoundBlaster.IRQ==2 || SoundBlaster.IRQ==10 || SoundBlaster.IRQ==11)
      outportb (0x21, inportb(0x21) & 251);
}

void RestoreIRQ (void)
/*------------------------------------------------------------------------
                                פונקציה לשחזור פסיקת ההשמעה המחזורית המקורית.
                                                          פרמטרים מועברים - אין.
                                                                 ערך מוחזר -  אין.
 ------------------------------------------------------------------------*/
```

```
{
   ///// 104 שחזר את הפסיקה המקורית ששמרנו בתא
   switch (SoundBlaster.IRQ)
   {
      case  2: setvect (0x71, getvect(104));   break;
      case 10: setvect (0x72, getvect(104));   break;
      case 11: setvect (0x73, getvect(104));   break;
      default: setvect (8+SoundBlaster.IRQ, getvect(104));
   }
   ///// IRQ-אפשר את ה
   switch (SoundBlaster.IRQ)
   {
      case  2: outportb (0xA1, inportb(0xA1) | 2);  break;
      case 10: outportb (0xA1, inportb(0xA1) | 4);  break;
      case 11: outportb (0xA1, inportb(0xA1) | 8);  break;
      default: outportb (0x21, inportb(0x21) | (1<<SoundBlaster.IRQ));
   }
   if (SoundBlaster.IRQ==2 || SoundBlaster.IRQ== 10 || SoundBlaster.IRQ==11)
      outportb (0x21, inportb(0x21) | 4);
}

int InitSoundBlaster (void)
/*------------------------------------------------------------------------
                                              פונקציה לאתחול כרטיס הקול.
                                                 פרמטרים מועברים - אין.
                                 ערך מוחזר -  אמת אם האתחול הסתיים בהצלחה.
 ------------------------------------------------------------------------*/
{
   if (!FindSoundBlaster())                 // מצא את כרטיס הקול
      return (0);                           // אם לא נמצא החזר שקר
   InitIRQ (ServiceIRQ);                    // אתחול תוכנית הפסיקה שלנו
   AssignBuffer ();                         // הקצאת זיכרון לחוצץ
   Mem.RBuffer = WAVStatus.Playing = 0;     // אתחול המשתנים

   return(1);                               // אתחול כרטיס הקול התבצע בהצלחה
}
void CloseSoundBlaster (void)
/*------------------------------------------------------------------------
   פונקציה ליציאה מהיחידה (שחרור הזיכרון שהוקצה ושחזור הפסיקה המקורית).
                                                 פרמטרים מועברים - אין.
                                                     ערך מוחזר -  אין.
 ------------------------------------------------------------------------*/
```

```
{
   if (WAVStatus.Playing) StopPlayback ();  // אם מנגן, עצור
   free (Mem.DMABuffer);                    // שחרור הזיכרון שהוקצה לחוצץ
   RestoreIRQ ();                           // שחזור פסיקת IRQ
}

int PlayWAV (char *FileName)
/*-----------------------------------------------------------------------
                                        פונקציה להשמעת קובץ WAV.
                                        פרמטרים מועברים :
                                        FileName - שם קובץ ה-WAV.
                                        ערך מוחזר :
                                        1 > ההשמעה הופעלה בהצלחה.
                                        -1 > אין מספיק זיכרון, או שכרטיס הקול לא אותחל.
                                        -2 > קובץ WAV אינו קיים.
                                        -3 > הקובץ אינו קובץ WAV תקין.
 -----------------------------------------------------------------------*/
{
   long Temp;
   if (WAVStatus.Playing) StopPlayback ();  // אם מנגן, עצור
   if (!Mem.DMABuffer)     // אם אין מספיק זיכרון או שכרטיס הקול לא אותחל
      return(-1);
   Mem.RBuffer = 0;                          // התחלת ההשמעה היא מאפס
   memset (Mem.DMABuffer, 0, 32768);         // אפס את החוצץ
   WAVStatus.WAVFile = fopen(FileName, "rb"); // פתח את הקובץ
   if (!WAVStatus.WAVFile) return (-2);      // צא אם לא קיים
   ///// קרא את בלוק ה-RIFF
   fread (&RIFFBlock, sizeof (RIFFBlock), 1, WAVStatus.WAVFile);
   if ( RIFFBlock.BlockType != 0x46464952 ||
       RIFFBlock.RIFF != 0x45564157)          // אם זהו לא קובץ WAV תקין
   {
      fclose (WAVStatus.WAVFile);
      return(-3);
   }
   ///// קריאת גוש פורמט ה-WAV
   do
   {
      Temp = ftell (WAVStatus.WAVFile);
      fread (&WAVFormatBlock,sizeof(WAVFormatBlock),1,WAVStatus.WAVFile);
      fseek (WAVStatus.WAVFile,Temp+WAVFormatBlock.BlockSize+8,SEEK_SET);
   } while (WAVFormatBlock.BlockType != 0x20746D66);
```

```
   ///// קביעת קצב הדגימה
   WriteDSP (0x40);
   WriteDSP (256 - 1000000 / WAVFormatBlock.SampleRate);
   ///// מציאת תחילת גוש הנתונים
   do
   {
      Temp = ftell (WAVStatus.WAVFile);
      fread (&BeginingOfDataBlock,sizeof(BeginingOfDataBlock),1,
            WAVStatus.WAVFile);
      if (BeginingOfDataBlock.BlockType != 0x61746164)
         fseek (WAVStatus.WAVFile,Temp+BeginingOfDataBlock.BlockSize + 8,
               SEEK_SET);
   } while (BeginingOfDataBlock.BlockType != 0x61746164);
   WAVStatus.ToBeRead=WAVStatus.ToBePlayed=BeginingOfDataBlock.BlockSize;
   ///// איפשור ה-DSP
   WriteDSP (0xD1);
   ///// קריאת החוצצים הראשון
   ReadBuffer (0);
   ReadBuffer (1);

   WAVStatus.Playing = 1;                  // מצב ההשמעה - מנגן
   if (WAVStatus.ToBeRead > 0)             // אם ישנו יותר ממחזור אחד
      InitAutoPlayback ();                 // הפעל השמעה אוטומטית
   else
      SingleCyclePlayback ();              // אחרת הפעל השמעת מחזור בודד
   return(1);
}

int WAVPlaying ()
/*-------------------------------------------------------------------------
      פונקציה הבודקת האם מתנגן כרגע קובץ כלשהו.
      פרמטרים מועברים - אין.
      ערך מוחזר - אמת אם כן.
 -------------------------------------------------------------------------*/
{
  return (WAVStatus.Playing);
}
```

תוכנית לדוגמה

התוכנית מדגימה השמעת קובץ WAV בעזרת היחידה שכתבנו.

בתחילת התוכנית מתבצע אתחול של כרטיס הקול. אם האתחול נכשל, מוצגת הודעה מתאימה. אם האתחול מתבצע בהצלחה, מושמע קובץ WAV.

לולאת while מחכה עד שייגמר השיר, או עד שהמשתמש יקיש על מקש כלשהו. ביציאה מהלולאה מוצגת הודעה האומרת, שההשמעה הסתיימה ויש להקיש על מקש כלשהו ליציאה.

בסוף התוכנית, אנו מכבים את כרטיס הקול, כדי שיפסיק להשמיע רעשים. אם המשתמש לוחץ על מקש לפני שההשמעה מסתיימת, פונקציית הכיבוי תדאג לעצור את ההשמעה לפני כיבוי כרטיס הקול.

הקבצים שיש לצרף לפרויקט כדי להריץ את הדוגמה הם: Wav.c ו-Wavtest.c. למידע נוסף קראו את נספח ו'.

```
/*** WAVTest.C ***/

#include <stdio.h>
#include "WAV.h"

void main (void)
{
   if (!InitSoundBlaster())               // בדיקת אתחול כרטיס הקול
   {
      printf("כרטיס הקול לא זוהה !");
      getch();
      exit(1);
   }
   PlayWAV ("Lion.wav");
   while (WAVPlaying() && !kbhit());
   printf("ההשמעה הסתיימה, לחצו על מקש כלשהו ליציאה");
   getch();
   CloseSoundBlaster();                   // כיבוי כרטיס הקול
}
```

15.8 רשימת פקודות DSP

סעיף זה מיועד לאלה, שרוצים לבנות יחידה לתמיכה בסוגים נוספים של קבצי WAV.

רשימה זו כוללת את פקודות DSP. עקבו שוב אחרי יחידת התכנות שבפרק זה. בכל פעם שנשלחת פקודה ל-DSP (בעזרת הפונקציה WriteDSP), חפשו את הפקודה ברשימת הפקודות ונסו למצוא את הפקודה שצריכה להחליף אותה עבור פורמט WAV שברצונכם להשמיע.

הערה

הערכים הנשלחים ל-DSP הם בגודל בית אחד (8 סיביות). לעיתים נרצה לשלוח ל-DSP, ערכים שגודלם עולה על שמונה סיביות. במקרים אלה, נפצל את הערך לחלקים של שמונה סיביות ונשלח אותם זה אחר זה. סדר השליחה של חלקים אלה משתנה בין הפקודות השונות, לכן יש לשים לב לרשימת השליחה.

לדוגמה: ניקח כדוגמה את הפקודה 48H (קביעת אורך מחזור השמעה מסוג 16KB).

תיאור הערכים לשליחה של פקודה זו:

ערכים לשליחה: 48H, שמונה הסיביות העליונות של ערך הקטן ב-1 מאורך המחזור, שמונה הסיביות התחתונות.

בפרק זה עבדנו עם מחזורים בגודל 16KB. ב-1KB יש 1024 בתים ולכן 16KB = 16384 בתים. ברשימת הערכים לשליחה כתוב "ערך הקטן ב-1 מאורך המחזור", לכן נפחית מערך זה אחד ונקבל 16383 בתים. עכשיו נמיר ערך זה לערך הקסדצימלי (ניתן לעשות זאת בעזרת המחשבון המדעי של Windows) ונקבל את הערך 3fffH. כל ספרה הקסדצימלית תופסת ארבע סיביות, לכן הערך שקיבלנו תופס 16 סיביות. ברשימת הערכים לשליחה, כתוב שיש לשלוח קודם את שמונת הסיביות העליונות ואחר כך את שמונת הסיביות התחתונות, לכן נעשה זאת כך:

```
WriteDSP (0x48);
WriteDSP (0xff);
WriteDSP (0x3f);
```

♫ פקודה 10H

פלט דגימה במוד גישה ישירה.

ערכים לשליחה: 10H, בית נתונים המכיל את הדגימה.

♫ פקודה 14H

קביעת אורך מחזור השמעה מסוג 8 סיביות.

ערכים לשליחה: 14H, שמונה הסיביות התחתונות של ערך הקטן ב-1 מהאורך, שמונה הסיביות העליונות.

♫ פקודה 1cH

אתחול השמעה מחזורית של 8 סיביות.

ערכים לשליחה: 1cH

הערה

יש להשתמש תחילה בפקודה 8H.

♫ פקודה 2cH

אתחול הקלטה מחזורית של 8 סיביות.

ערכים לשליחה: 2cH

הערה

יש להשתמש תחילה בפקודה 48H.

♫ פקודה 20H

קליטת דגימה במוד גישה ישירה.

ערכים לשליחה: 20H

ערכים לקריאה: דגימה בגודל בית.

♫ פקודה 24H

קביעת גודל מחזור הקלטה בודד של 8 סיביות.

ערכים לשליחה: 24H, שמונה הסיביות התחתונות של ערך הקטן ב-1 מהאורך, שמונה הסיביות העליונות.

♫ פקודה 40H

קביעת קצב הדגימה.

ערכים לשליחה: X,40H.

x = (65536 - (256000000/מספר הערוצים*קצב הדגימה)) / 256

♫ פקודה 41H

קביעת תדר ההשמעה.

ערכים לשליחה: 41H, שמונה הסיביות העליונות של התדר בהרץ, שמונה הסיביות התחתונות.

פקודה 42H

קביעת תדר ההקלטה.

ערכים לשליחה: 42H, שמונה הסיביות העליונות של התדר בהרץ, שמונה הסיביות התחתונות.

פקודה 48H

קביעת אורך המחזור להשמעה מחזורית.

ערכים לשליחה: 48H, שמונה הסיביות העליונות של ערך הקטן ב-1 מאורך המחזור, שמונה הסיביות התחתונות.

פקודה 80H

השהיית ההקלטה.

ערכים לשליחה: 80H, שמונה הסיביות התחתונות של ערך הקטן ב-1 מזמן ההשהיה הרצוי, שמונה הסיביות העליונות.

פקודה 90H

אתחול השמעה רציפה ומהירה מסוג 8 סיביות (השמעה במהירות גדולה מהרגיל).

ערכים לשליחה: 90H.

הערה

יש להשתמש קודם בפקודה 48H.

פקודה 91H

אתחול השמעה רציפה ומהירה של מחזור בודד מסוג 8 סיביות (השמעה במהירות גדולה מהרגיל).

ערכים לשליחה: 91H.

הערה

יש להשתמש קודם בפקודה 48H.

פקודה 98H

אתחול הקלטה מהירה מחזורית מסוג 8 סיביות.

ערכים לשליחה: 98H.

הערה

יש להשתמש קודם בפקודה 48H.

♫ פקודה 99H

אתחול הקלטה מהירה של מחזור בודד מסוג.

ערכים לשליחה: 99H.

הערה: **יש להשתמש קודם בפקודה 48H.**

♫ פקודה a0H

קביעת הקלטה מסוג מונו.

ערכים לשליחה : a0H.

♫ פקודה a8H

קביעת הקלטה מסוג סטריאו.

ערכים לשליחה : a8H.

♫ פקודות b?H

פקודות להשמעה והקלטה מסוג 16 סיביות.

ערכים לשליחה : b?H, מוד, שמונה הסיביות התחתונות של ערך הקטן ב-1 מכמות הדגימות שבמחזור, שמונה הסיביות העליונות.

? - ספרה אחת (4 סיביות) הנקבעת על פי הצירוף הבא :

שתי הסיביות הראשונות (b1,b0) תמיד אפס.

סיבית שלישית (b2) תהיה אפס למחזור בודד ו-1 להשמעה מחזורית.

סיבית רביעית (b3) תהיה אפס להשמעה ו-1 להקלטה.

מוד - משתנה מסוג בית שערכו יהיה אפס למונו ו-20H לסטריאו.

♫ פקודה d0H

השהיית שידור DMA מסוג 8 סיביות.

ערכים לשליחה : d0H.

♫ פקודה d1H

הדלקת ה-DSP.

ערכים לשליחה : d1H.

♫ פקודה d3H

כיבוי ה-DSP.

ערכים לשליחה : d3H.

♫ פקודה d4H

חזרה מהשהיית שידור DMA מסוג 8 סיביות.

ערכים לשליחה : d4H.

♫ פקודה d5H

עצירת שידור DMA מסוג 16 סיביות.

ערכים לשליחה : d5H.

♫ פקודה d6H

חזרה מהשהיית השהיית שידור DMA מסוג 16 סיביות.

ערכים לשליחה : d6H.

♫ פקודה d8H

קביעת סטטוס ה-DSP.

ערכים לשליחה : d8H.

ערכים לקריאה : הסטטוס (00 - כבוי, ffH - דלוק).

♫ פקודה d9H

עצור השמעה מחזורית מסוג 16 סיביות בהפעלה הבאה של פסיקת השירות.

ערכים לשליחה : d9H.

♫ פקודה daH

עצור השמעה מחזורית מסוג 8 סיביות בהפעלה הבאה של פסיקת השירות.

ערכים לשליחה : daH.

פרק 16

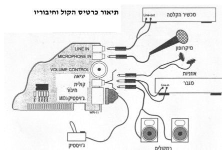
הקלטת קבצי WAV

למדנו להשמיע שירים מכונן התקליטורים וגם להשמיע קבצי קול מסוג WAV. להשלמת נושא פרקי השמע (sound), נלמד בפרק זה להקליט שירים מכונן התקליטורים לשמירה כקבצי קול מסוג WAV.

אם אינכם רוצים להקליט שירים מכונן התקליטורים, תוכלו להקליט את עצמכם בעזרת המיקרופון, וכך לשלב בתוכנית, או בפרויקט כלשהו גם הסברים וקולות אחרים.

אפשרות נוספת היא לחבר לכרטיס הקול מכשיר חיצוני, כמו אורגנית, דרך הכניסה המסומנת בו : **Line In** או **Line-In**.

תיאור כרטיס הקול וחיבוריו



הערה

כדי להפיק את מלוא התועלת מלימוד פרק זה, רצוי להכיר תחילה את הנושאים המפורטים בפרק 15, "השמעת קבצי WAV".

16.1 המיקסר - Mixer

בפרק הקודם הכרנו את הרכיב **DSP** (מעבד אותות דיגיטליים), שנמצא על כרטיס הקול. עכשיו נכיר רכיב נוסף בכרטיס הקול – **Mixer** (מערבל, מיקסר).

המיקסר מורכב מאוגרים רבים, שכל אחד מהם בעל תפקיד נפרד, כמו למשל: עוצמת הקול בהשמעה מכונן התקליטורים, עוצמת הקול בהקלטה מהמיקרופון, קביעת הרכיב המחובר כמקור קלט לצלילים ועוד.

כדי לשנות ערך של אחד האוגרים שבמיקסר, עלינו לשלוח את קוד האוגר המבוקש אל היציאה הנמצאת בהיסט 4 מכתובת הבסיס, ואחר כך לשלוח את הערך הרצוי ליציאה הנמצאת בהיסט 5 מכתובת הבסיס.

הפונקציה תיראה כך:

```
void WriteMixer(byte R, byte V)
{
 outportb (SoundBlaster.Base + 4, R);
 outportb (SoundBlaster.Base + 5, V);
}
```

אם נרצה לבדוק מהו ערך של אוגר מסוים במיקסר, נשלח את קוד האוגר ליציאה הנמצאת בכתובת הבסיס בהיסט 4, ומייד אחר כך נקרא את ערך האוגר מהיציאה הנמצאת בכתובת הבסיס בהיסט 5.

הפונקציה תיראה כך:

```
byte ReadMixer(byte R)
{
  outportb (SoundBlaster.Base + 4, R);
  return ( inportb(SoundBlaster.Base + 5) );
}
```

לפני שנתחיל בהקלטה עלינו להיעזר במיקסר כדי לכוון את מקור ההקלטה (כונן התקליטורים, מיקרופון או חיבור Line In), לכוון את עוצמת ההקלטה לעוצמה מקסימלית ולבטל את ערוץ ההקלטה הימני (כי אנו עוסקים בקבצי קול מסוג מונו).

נגדיר שלושה קבועים, שיסמלו את מקורות ההקלטה האפשריים. הערכים של הקבועים אינם חשובים (חשוב שיהיו ערכים שונים, כדי שנוכל להבדיל ביניהם).

```
#define CD 1                          // הקלטה מכונן התקליטורים
#define MIC 2                         // הקלטה מהמיקרופון
#define LINEIN 3                      // Line-In הקלטה מחיבור
```

עכשיו נכתוב את הפונקציה שתקבל קבוע של אחד המקורות, ותבקש מהמיקסר לחבר אותו כמקור. אין צורך להתעמק בנושא המיקסר ובמשמעות האוגרים וערכיהם (במיוחד לאור העובדה, שזו הפונקציה היחידה בספר זה בה נעזר במיקסר), אלא רק להבין את השימוש בהם.

```
int SelectSource(char Source)
/*------------------------------------------------------------------------
                        פונקציה לבחירת מקור ההקלטה ועדכון ה-Mixer בהתאם.
                        פרמטר מועבר - קבוע מקור ההקלטה (LINE-IN,MIC,CD).
                        ערך מוחזר -  אמת בהצלחה.
 ------------------------------------------------------------------------*/
{
                                             ביטול ערוץ ההקלטה הימני //
   WriteMixer (0x3E, 0);

          חיבור הקלטה לערוץ השמאלי של המקור הרצוי בעוצמת קול מקסימלית //
   switch (Source)
   {
     case MIC    :
      {
         WriteMixer (0x3D, 1);
         WriteMixer (0x0A, 7);
         WriteMixer (0x3C, 0);
         break;
      }
     case CD     :
      {
         WriteMixer (0x3D, 6);
         WriteMixer (0x28, 255);
         WriteMixer (0x3C, 6);
         break;
      }
     case LINEIN :
      {
         WriteMixer (0x3D, 24);
         WriteMixer (0x2E, 255);
         WriteMixer (0x3C, 24);
         break;
      }
     default     : return(0);
   }
   return(1);
}
```

16.2 מנגנון ההקלטה המחזורית

הסקרנים מביניכם שהציצו ברשימת פקודות ה-DSP שבפרק הקודם, בוודאי שמו לב שקיימות פקודות הקלטה המקבילות לפקודות ההשמעה המחזורית, בהם השתמשנו בפרק הקודם. אם כך, עלינו לנסות ולגלות מה צריך לשנות בפונקציות שכתבנו להשמעת קבצי WAV, כדי שיתאימו לביצוע הקלטה.

לפני שניגש לפונקציות עצמן, נסו לחשוב האם חסרים לנו משתנים כלשהם ?

כדי לעקוב אחר התקדמות ההשמעה, השתמשנו במשתנים ToBePlayed ו-ToBeRead הנמצאים בתוך המבנה WAVStatus. כדי לעקוב אחרי התקדמות ההקלטה, נשתמש במשתנה Recorded, שיכיל את מספר הדגימות שהוקלטו. נוסיף אותו למבנה:

```
struct {                          // מבנה לשמירת סטטוס השמעת WAV
 FILE *WAVFile ;                  // מצביע אל קובץ WAV
  int Playing ;                   // האם הקובץ בהשמעה עכשיו?
  long ToBeRead ;                 // מספר הדגימות שנשארו לקרוא מהקובץ
  long ToBePlayed ;               // מספר הדגימות שנשארו להשמיע
  long Recorded ;                 // מספר הדגימות שהוקלטו
  } WAVStatus ;
```

כדי לעקוב אחר מחצית החוצץ שממנו אנו קוראים, השתמשנו במשתנה Rbuffer, הנמצא במבנה Mem. כדי לעקוב אחרי מחצית החוצץ שאליו נכתוב, נוסיף למבנה את המשתנה Wbuffer. המבנה המעודכן ייראה כך:

```
struct {                          // מבנה לשמירת נתוני הזיכרון הדרושים
 short Page ;                     // דף זיכרון
 short Offset;                    // היסט
 byte *DMABuffer;                 // מצביע לחוצץ DMA
 short RBuffer;                   // מיקום הקריאה בחוצץ
 short WBuffer;                   // מיקום הכתיבה בחוצץ
 } Mem;
```

עכשיו כל המשתנים מוכנים ואפשר לעבור לטיפול בפונקציות.

נתחיל בפונקציה ReadBuffer. כזכור, פונקציה זו טוענת נתונים מקובץ WAV אל מחצית קיבולת החוצץ. הפעם, כיון שאנו עוסקים בהקלטה, עלינו לכתוב פונקציה נוספת שתבצע פעולה הפוכה ותשמור לתוך קובץ WAV את מחצית תוכן החוצץ. הפונקציה תיראה כך:

```
void SaveBuffer (short Buffer)
{
   fwrite (Mem.DMABuffer + (Buffer << 14), 16384, 1, WAVStatus.WAVFile);
   WAVStatus.Recorded += 16384;          // עדכן את מונה הדגימות שהושמעו
}
```

במנגנון ההשמעה המחזורי, תוכנית השירות של בקשת הפסיקה עסקה בקריאת הקובץ אל מחצית החוצץ. לכן, עלינו לשנות תוכנית זו, כך שבמקום להפעיל את הפונקציה ReadBuffer בסוף כל מחזור, התוכנית תפעיל את הפונקציה WriteBuffer. בנוסף, נחליף את המעקב אחר מוני התקדמות ההשמעה במעקב אחרי התקדמות ההקלטה (Recorded).

תוכנית הפסיקה ה"משופצת" תיראה כך (שימו לב לשינויים):

```
void interrupt RecordingServiceIRQ (void)
{
                                          // שחרר את ה-DSP
   inportb (0x22E);
                                          // הכר בפסיקת החומרה
   outportb (0x20, 0x20);
                                  // הכר בפסיקה הגולשת (עבור IRQ ו 10 ,2-11)
   if (SoundBlaster.IRQ==2 || SoundBlaster.IRQ==10 || SoundBlaster.IRQ==10)
      outportb (0xA0, 0x20);
                                          // שמירת החוצץ
   SaveBuffer (Mem.WBuffer);
                                          // עבור למחצית החוצץ הבאה
   Mem.WBuffer ^= 1;
}
```

כדי לטעון אל וקטור הפסיקות את הפסיקה החדשה שכתבנו, נכתוב פונקציה לאתחול כרטיס הקול להקלטה. פונקציה זו תהיה זהה לפונקציה InitSoundBlaster, אלא שעכשיו היא טוענת את תוכנית הפסיקה להקלטה ולא את תוכנית הפסיקה להשמעה. כמו כן, היא גם מאפסת את מונה התקדמות ההקלטה, ולא את מונה התקדמות ההשמעה.

הפונקציה תיראה כך:

```
int InitSoundBlasterForRecording (void)
{
   if (!FindSoundBlaster())               // מצא את כרטיס הקול
      return (0);                         // אם לא נמצא החזר שקר
   InitIRQ (RecordingServiceIRQ);         // אתחול ה-IRQ בפסיקה שלנו
   AssignBuffer ();                       // הקצאת זיכרון לחוצץ
   Mem.WBuffer = WAVStatus.Recorded = 0;  // אתחול המשתנים

   return(1);                             // אתחול כרטיס הקול התבצע בהצלחה
}
```

לאחר שאנו יודעים כיצד מבוצעת הקלטה, נוכל לאתחל הקלטה מחזורית. אתחול הקלטה המחזורית זהה לאתחול ההשמעה המחזורית (InitAutoPlayback), מלבד שני שינויים קטנים:

1. עלינו להגדיר את ערוץ ה-DMA לכתיבה, ולא לקריאה.
2. בסוף הפונקציה, נחליף את פקודת ה-DSP הקיימת להשמעה מחזורית בפקודה אחרת – להקלטה מחזורית.

מלבד שני שינויים אלה (שמסומנים בקו תחתי), הפונקציות זהות לחלוטין.

```
void StartRecording (void)
{
   ///// הכן את ערוץ DMA להקלטה מחזורית אוטומטית
   outportb (0x0A, 4 | SoundBlaster.DMA);
   outportb (0x0C, 0);
   outportb (0x0B, 0x54 | SoundBlaster.DMA);
   ///// שליחת ההיסט לבקר ה-DMA
   outportb (SoundBlaster.DMA << 1, Mem.Offset & 0xFF);
   outportb (SoundBlaster.DMA << 1, Mem.Offset >> 8);
   ///// כתיבת הדף אל בקר DMA
   switch (SoundBlaster.DMA)
   {
      case 0 : outportb (0x87, Mem.Page);  break;
      case 1 : outportb (0x83, Mem.Page);  break;
      case 3 : outportb (0x82, Mem.Page);  break;
   }
   ///// קביעת אורך הבלוק של ה-DMA ל-7fffH (32KB)
   outportb ((SoundBlaster.DMA << 1) + 1, 0xFF);
   outportb ((SoundBlaster.DMA << 1) + 1, 0x7F);
   outportb (0x0A, SoundBlaster.DMA);
   ///// קביעה שאורך הבלוק של ה-DSP יהיה 3fffH (16KB)
   WriteDSP (0x48);
   WriteDSP (0xFF);
   WriteDSP (0x3F);
   ///// התחל את ההקלטה המחזורית
   WriteDSP (0x2C);
}
```

ובכן, הפעלנו את ההקלטה המחזורית ואנו יודעים כיצד היא מתבצעת, אבל שכחנו דבר חשוב מאוד – לקבוע את תדר ההקלטה ולהכין את הקובץ להקלטה.

לשם כך, נכתוב פונקציה כללית להפעלת ההקלטה. הפונקציה תקבל כפרמטרים את שם הקובץ שלתוכו אנו רוצים להקליט, וקבוע המציין את המקור שממנו אנו רוצים להקליט.

הפונקציה צריכה לבצע פעולות אלו:

1. לקשר את המקור הרצוי (כגון התקליטורים, מיקרופון וכו') ככלט להקלטה (בעזרת הפונקציה SelectSource שיצרנו).
2. לפתוח את הקובץ לכתיבה.
3. להכניס לקובץ כותר (הזהה לכותר שהכרנו בפרק הקודם).
4. לקבוע את תדר ההקלטה (בדיוק כפי שקבענו את תדר ההשמעה).
5. להפעיל את ההקלטה המחזורית (בעזרת הפונקציה StartRecording, שיצרנו כרגע).

עד לסיום ההקלטה, אין אנו יודעים מהם הנתונים המדויקים שכותר הקובץ צריך להכיל. בינתיים, נכניס כותר מדומה (ריק), ובסוף ההקלטה נעדכן אותו בנתונים האמיתיים. הכותר המינימלי ביותר שיכול להיות לקובץ WAV, מכיל את שלושת הבלוקים שהגדרנו בפרק הקודם (RIFFBlock, WAVFormatBlock ו- BeginingOfDataBlock). עבור שלושה בלוקים אלה דרושים 44 בתים, לכן נוסיף לקובץ 44 בתים ריקים.

הפונקציה תיראה כך:

```
int RecordWAV (char *FileName, char Source)
{
   byte Empty=0,i;

   if (!SelectSource(Source))                         // אתחול מקור ההקלטה
      return (0);                                     // צא אם המקור שגוי
   WAVStatus.WAVFile = fopen (FileName, "wb");        // פתח את הקובץ לכתיבה
   if (!WAVStatus.WAVFile)                            // צא אם הפתיחה נכשלה
      return (0);

    ///// הכנסת 44 בתים ריקים לקובץ (במקום הכותר הנכון שיעודכן בסוף ההקלטה) /////
    for (i=0;i<44;i++)
      fwrite (&Empty, 1, 1, WAVStatus.WAVFile);
    ///// קביעת תדירות ההקלטה /////
    WriteDSP (0x40);
    WriteDSP (165);
    WAVStatus.Recorded = 0;                           // איפוס מונה ההקלטה
    StartRecording ();                                // התחל את ההקלטה
    return (1);
}
```

16.3 תעצרו את הרכבת, אני רוצה לרדת

מנגנון ההקלטה המחזורית האוטומטית כבר מוכן, אך כיצד נוכל לעצור אותו?

כדי לעצור את ההשמעה המחזורית, תוכנית הפסיקה צריכה לגלות מתי אנו נמצאים במחזור שלפני האחרון, ואם זה המצב, עלינו להפעיל את הפקודה daH (עצירת ההשמעה המחזורית במחזור הבא) של מעבד האותות DSP. הפונקציה לקריאת חצי חוצץ מהקובץ זיהתה את המחזור האחרון ומילאה את שארית החוצץ ב"דממה".

עצירת ההקלטה המחזורית אינה יכולה להתבצע מייד, כשם שאי אפשר לעצור מייד את ההשמעה. לשם כך נפעיל את הפקודה daH של ה-DSP. כדי שלא נמשיך להקליט במשך שארית המחזור לאחר שהמשתמש ביקש לעצור את ההקלטה, נפעיל את הפקודה d0H, הגורמת להשהיה. כך נוכל להשהות את ההקלטה ברגע שהמשתמש מבקש לעצור אותה, ומייד בתחילת המחזור הבא להפוך זאת לעצירה מוחלטת של המנגנון המחזורי.

בנוסף לעצירת ההקלטה, הפונקציה לסיום הקלטה צריכה גם לכתוב בכותר המדומה של הקובץ את הנתונים האמיתיים הנובעים מההקלטה. כדי לדעת מהו אורך הקובץ המוקלט, נוסיף למונה הדגימות שהוקלטו, ערך המציין את מספר הדגימות שהוקלטו במחזור זה, עד שהופעלה הפונקציה (מיקומו של מצביע DMA ביחס לגודל החוצץ).

הפונקציה נראית כך:

```
void EndRecording (void)
{
   short Counter, DMAPointer;
   ///// עצירת השידור
   WriteDSP (0xD0);
   WriteDSP (0xDA);
   ///// קרא את מצביע ה-DMA מבקר ה-DMA
   DMAPointer = inportb (1 + (SoundBlaster.DMA >> 1));
   DMAPointer += (inportb (1 + (SoundBlaster.DMA >> 1)) << 8);
   ///// כמות הדגימות שהוקלטה מתחילת המחזור הנוכחי
   Mem.Offset = 0x7FFF - DMAPointer;
   ///// שמירת הדגימות שהוקלטו מתחילת המחזור הנוכחי
   fwrite (Mem.DMABuffer + (Mem.Offset & 16384),Mem.Offset & 16383,
        1, WAVStatus.WAVFile);
   ///// הוספת כמות דגימות זו למונה הדגימות שהוקלטו
   WAVStatus.Recorded += Mem.Offset & 16383;

   ///// קביעת ערכי בלוק RIFF
   RIFFBlock.BlockType = 0x46464952;
   RIFFBlock.BlockSize = WAVStatus.Recorded + 36;
   RIFFBlock.RIFF = 0x45564157;
```

```
    ///// קביעת ערכי בלוק פורמט WAV /////
    WAVFormatBlock.BlockType = 0x20746D66;
    WAVFormatBlock.BlockSize = 16;
    WAVFormatBlock.WaveType = 1;
    WAVFormatBlock.Channels = 1;                   // מונו
    WAVFormatBlock.SampleRate = 11025;             // 11KHz
    WAVFormatBlock.BytesPerSecond = WAVFormatBlock.Channels *
                                          WAVFormatBlock.SampleRate;
    WAVFormatBlock.BlockAlignment = 1;
    WAVFormatBlock.BitResolution = 8;

    ///// קביעת ערכי תחילת בלוק הנתונים /////
    BeginingOfDataBlock.BlockType = 0x61746164;
    BeginingOfDataBlock.BlockSize = WAVStatus.Recorded;

    ///// עדכון הבלוקים בקובץ /////
    fseek (WAVStatus.WAVFile, 0, SEEK_SET);
    fwrite (&RIFFBlock, sizeof(RIFFBlock), 1, WAVStatus.WAVFile);
    fwrite (&WAVFormatBlock, sizeof(WAVFormatBlock), 1, WAVStatus.WAVFile);
    fwrite (&BeginingOfDataBlock, sizeof(BeginingOfDataBlock), 1,
                                    WAVStatus.WAVFile);
    fclose (WAVStatus.WAVFile);
}
```

16.4 היחידה לקבצי WAV

להקלטת קבצי WAV, משתמשים ברוב הפונקציות שכתבנו ביחידה להשמעת קבצי WAV (ראו פרק קודם, פרק 15). עכשיו ניצור יחידה WAV אחת, שתשלב את ההשמעה ואת ההקלטה. לשם כך, נוסיף ליחידה של הפרק הקודם את כל הפונקציות ואת השינויים שהוזכרו בפרק זה. היחידה המלאה נמצאת בתיקיה של פרק זה (תחת התיקיה books\59281).

תוכנית לדוגמה

התוכנית לדוגמה מדגימה הקלטה מכונן התקליטורים. ההקלטה מתחילה ומופסקת בלחיצה על מקש כלשהו. הקובץ המוקלט נקרא MyWAV.WAV.

הקבצים שיש לצרף לפרויקט כדי להריץ את הדוגמה הם: Wav.c ו-Recest.c. למידע נוסף קראו את נספח ו'.

```
/*** RecTest.C ***/

#include <stdio.h>
#include "wav.h"

void main (void)
{
   if (!InitSoundBlasterForRecording())
   {
      printf ("אתחול כרטיס הקול נכשל");
      exit(1);
   }
   printf ("כרטיס הקול אותחל בהצלחה, הקש מקש כלשהו כדי להתחיל בהקלטה\n");
   getch();
   RecordWAV("MyWAV.WAV",CD);
   printf ("ההקלטה החלה, לחץ על מקש כלשהו לסיום ההקלטה\n");
   while (!kbhit());
   getch();
   EndRecording();
   CloseSoundBlaster();
}
```

פרק 17

תקשורת טורית בין מחשבים והפעלת המודם

בפרק זה נעסוק בתקשורת טורית בין שני מחשבים, המחוברים באמצעות כבל טורי או באמצעות קו טלפון ומודם.

- ✓ **נלמד לאתחל את יציאת התקשורת (COM).**
- ✓ **נלמד ליצור תוכנת צ'אט בין שני מחשבים, המחוברים בכבל טורי.**
- ✓ **נלמד להתגבר על שגיאות בשידור.**
- ✓ **נלמד ליצור חייגן טלפון.**
- ✓ **נלמד להעביר קבצים בין שני מחשבים בעזרת כבל ובעזרת מודם.**

17.1 תקשורת טורית

תקשורת טורית היא תקשורת אסינכרונית המאפשרת למחשב לתקשר עם התקנים חיצוניים (כגון עכבר), או עם מחשבים אחרים המחוברים אליו, בעזרת כבל או מודם וקו טלפון.

בשידור סינכרוני יש סינכרון (תיאום זמנים) תמידי בין המחשבים, המתבצע בדרך כלל על ידי הוספת קו סינכרון שמשודר בו אות שעון (דפקי מתח קצרים, המשודרים בכל קטע של שידור סיביות), המתאם בין השעונים (למעשה, קוצבי זמן) של שני המחשבים.

לעומתו, בשידור אסינכרוני, הסינכרון בין המחשבים (או ההתקנים), נשמר רק במסגרת של תו יחיד. כשלא משודר תו כלשהו, הקו נמצא במצב אדיש (**Idle**). לפני שידור תו כלשהו, משודרת סיבית התחלה (**start bit**), המסנכרנת בין המחשבים ומסמלת לתחנה המקבלת שהתחנה המשדרת עומדת לשדר תו. אחרי שידור סיבית ההתחלה משודרות סיביות התו, ואחריהן משודרת סיביות עצירה (**stop bit**). כאשר השולח משדר רצף של תווים, המרווחים בין התווים העוקבים, יהיו כולם זהים ושווים למרווח העצירה שנקבע. אם התווים אינם משודרים ברצף ויש הפסקות שרירותיות בין תו לתו, מרווח העצירה יימשך על ידי המצב האדיש.

אוסף הסיביות, הכולל: סיבית ההתחלה, סיביות המידע של התו, סיבית הבדיקה וסיביות העצירה, נקרא מסגרת (**Frame**).

היתרון של השידור האסינכרוני הוא בהיותו פשוט וזול למימוש, כי הסינכרון בין התחנות צריך להישמר רק במשך קטעי זמן קצרים (של שידור מסגרת תו), לכן הוא יכול להיות פחות מדויק ואינו מצריך אות שעון. החיסרון של השידור האסינכרוני הוא התקורה (**overhead**) הגבוהה. התקורה מבטאת את מספר הסיביות, שאינן סיביות מידע שיש להוסיף בכל מסגרת. במקרה זה היא כוללת את סיביות ההתחלה, העצירה וסיבית בדיקה (parity bit).

בכל מחשב מותקן כרטיס תקשורת (אחד לפחות), המאפשר תקשורת טורית חיצונית (חפשו מחבר זכר בצורת טרפז בעל 9 או 25 פינים בצידו האחורי של המחשב). לכל כניסה טורית יש שם ייחודי: הראשונה נקראת COM1, השנייה נקראת COM2 וכו'.

בנוסף לערוץ התקשורת (הכבל או קו הטלפון) ולכרטיס התקשורת, כדי ליישם תקשורת טורית בין מחשבים, דרושה גם תוכנית תקשורת המתווכת בין המשתמש לבין כרטיס התקשורת ומשמשת כפרוטוקול (אוסף כללים בין בצדדים המעורבים בתקשורת, שתפקידם להסדיר את העברת הנתונים ביניהם). את תוכנית התקשורת נבנה בעזרת שימוש בפסיקה 14H של BIOS, המספקת שירותים להפעלת כרטיס התקשורת. נעשה זאת כפי שלמדנו בפרק 3.

17.2 אתחול כרטיס התקשורת

כדי שכרטיס התקשורת יוכל לבנות מסגרת עבור כל תו שברצוננו לשדר, וגם לשדר אותה בערוץ התקשורת בקצב הרצוי, צריך לאתחל אותו תחילה.

בעת האתחול צריך למסור לכרטיס התקשורת פרמטרים אלה:

1. **מהירות השידור.** הקצב בו ישודרו הסיביות בערוץ התקשורת.
2. **מספר סיביות המידע במסגרת .** ניתן לשדר 7 סיביות נתונים עבור ASCII תקני, או 8 סיביות נתונים עבור ASCII מורחב (למידע נוסף אודות ASCII, ראה פרק 5).
3. **סוג הבדיקה.** האם תתוסף למסגרת סיבית זוגיות זוגית (Even Parity), או סיבית זוגיות אי-זוגית (Odd Parity), או שהמסגרת תשודר ללא סיבית בדיקה (הסבר מפורט אודות סיבית בדיקת הזוגיות, ראו סעיף 2.7 בפרק 2).
4. **מספר סיביות העצירה.** ניתן לקבוע סיבית עצירה אחת או שתיים.

הערה

בשידור אסינכרוני, תמיד משודרת בתחילת המסגרת סיבית התחלה אחת.

את ארבעת הפרמטרים האלה נכניס למשתנה בגודל בית, באופן הבא:

1. סיביות 0-1 יסמנו את מספר סיביות הנתונים (7 סיביות או 8).
2. סיבית 2 תסמן את מספר סיביות העצירה (סיבית עצירה אחת או שתיים).
3. סיביות 3-4 יסמנו את סוג הבדיקה (זוגיות זוגית, אי-זוגית, או ללא בדיקה).
4. סיביות 5-7 יסמנו את מהירות השידור בסל״ש (סיביות לשנייה).

b7	b6	b5	b4	b3	b2	b1	b0
מהירות			בדיקה		עצירה	מידע	
0 0 0 - 110 סל"ש 1 0 0 - 150 סל"ש 0 1 0 - 300 סל"ש 1 1 0 - 600 סל"ש 0 0 1 - 1200 סל"ש 1 0 1 - 2400 סל"ש 0 1 1 - 4800 סל"ש 1 1 1 - 9600 סל"ש			0 0 - ללא בדיקה 1 0 - זוגיות אי זוגית 1 1 - זוגיות זוגית		0 - אחת 1 - שתיים	0 1 - 7 סיביות 1 1 - 8 סיביות	

תרשים 17.1: תיאור בית מאפייני התקשורת

את האתחול נבצע בעזרת שירות מספר 0 של פסיקת התקשורת הטורית 14H.

לפני הפעלת הפסיקה:

אוגר ah – צריך להכיל 0 (מספר השירות).

אוגר al – צריך להכיל את בית מאפייני התקשורת.

אוגר dx – צריך להכיל את מספר החיבור (0 עבור COM1 או 1 עבור COM2).

את הפסיקה נפעיל בעזרת ()int86, כפי שלמדנו בפרק 3.

הערה

האתחול חייב להתבצע עם אותם מאפייני תקשורת בשני צידי הקישור (בשני המחשבים או ההתקנים המחוברים), כדי שהתקשורת תצליח.

17.3 בדיקת סטטוס התקשורת

שירות מספר 3 של פסיקת התקשורת הטורית מאפשר לקבל מידע אודות סטטוס התקשורת.

לפני הפעלת הפסיקה:

אוגר ah - צריך להכיל 3 (מספר השירות).

אוגר dx - צריך להכיל את מספר החיבור (0 עבור COM1, או 1 עבור COM2).

לאחר הפעלת הפסיקה, מוחזר סטטוס התקשורת באוגרים ah ו-al.

משמעות הסיביות השונות באוגר ah, כשהן במצב "**1**":

b0 – כרטיס התקשורת קלט תו ממחשב מרוחק (בהמשך נלמד כיצד לקלוט אותו לתוכנית).

b1 – ארעה תקלה, וכרטיס התקשורת קלט יותר מתו אחד. מכיון שבכרטיס יש חוצץ עבור תו אחד בלבד, שגיאה זו מסמלת שתו אחד לפחות אבד.

b2 – בדיקת הזוגיות גילתה שגיאה במסגרת שהתקבלה.

b3 – מסגרת הנתונים משובשת (סביר להניח שאין תיאום בין המהירויות של שתי התחנות).

b4 – כרטיס התקשורת קלט את התו Break, המסמל ניתוק זמני בתקשורת.

b5 – כרטיס התקשורת אינו משדר כרגע, וניתן לשדר דרכו מסגרת חדשה.

b6 – כרטיס התקשורת נמצא במצב אדיש (אינו פעיל).

b7 – התקשורת אינה תקינה (החיבורים אינם תקינים, או שהמחשב או ההתקן המחובר כבוי).

הערה

כדי לבדוק את ערכה של סיבית מסוימת נשתמש במסכות, כפי שלמדנו בפרק 2, "עבודה בסיביות".

שירות בדיקת סטטוס התקשורת מחזיר גם מידע נוסף באוגר al ומתייחס למודם, על כך נרחיב בהמשך כשנלמד על המודם. כפי שראיתם בתיאור מבנה הסטטוס המוחזר באוגר ah, אם הסיבית האחרונה בו מורמת, התקשורת אינה תקינה. אם כך, רצוי שלאחר ביצוע האתחול נבדוק את ערך סיבית זו, כך נדע אם האתחול בוצע בהצלחה.

פונקציית אתחול עם בדיקה תיראה כך:

```
int InitCom (int Com, int Speed)
{
   union REGS ireg;
   byte Properties;                 // בית מאפייני השידור

   switch (Speed)                   // קביעת מהירות החיבור
   {
      case 110 : Properties = 0;        break;
      case 150 : Properties = 1;        break;
      case 300 : Properties = 2;        break;
      case 600 : Properties = 3;        break;
      case 1200: Properties = 4;        break;
      case 2400: Properties = 5;        break;
      case 4800: Properties = 6;        break;
      case 9600: Properties = 7;        break;
      default  : return (0);
   }
   Properties = Properties << 5; //הזזת ערך המהירות שיתאים למבנה המאפיינים
   Properties += 0x1f;              // בדיקת זוגיות ושתי סיביות עצירה

   if ( Com==1 || Com==2 )
      ireg.x.dx = (--Com);          // הגדרת היציאה
   else
      return (0);
   ireg.h.ah = 0;                   // 0 - אתחול התקשורת
   ireg.h.al = Properties;

   int86 (0x14,&ireg,&ireg);        // הפעלת פסיקת התקשורת

   ireg.h.ah = 3;                   // 3 - בדיקת סטטוס
   int86 (0x14,&ireg,&ireg);        // הפעלת פסיקת התקשורת
   if (ireg.h.ah & 0x8)             // אם התרחשה תקלה
      return (0);
   else
      return (1);
}
```

17.4 שידור תו

כדי לשדר תו, עלינו להעביר לכרטיס התקשורת את הסיביות המרכיבות אותו ולבקש ממנו לשדר אותן. נעשה זאת בעזרת שירות מספר 1 של פסיקת התקשורת הטורית (14H).

לפני הפעלת הפסיקה:

אוגר ah - צריך להכיל 1 (מספר השירות).

אוגר al - צריך להכיל את התו שברצוננו לשדר.

ואוגר dx - צריך להכיל את מספר החיבור (0 עבור COM1 או 1 עבור COM2).

הערה

אם התחנה כבר משדרת מסגרת כלשהי בעת הפעלת שירות 1, בקשת השידור תתבטל והמסגרת החדשה לא תשודר. כדי למנוע זאת, עלינו לבדוק את הסטטוס לפני השידור. אם סיבית 5 אינה מורמת (התחנה משדרת כרגע), נחכה עד שהסיבית תרומם ורק אז נפעיל את שירות 1.



כתבו פונקציה בשם SendChar, המקבלת מספר יציאה (COM) ותו, ומשדרת את התו.

17.5 קליטת תו

כאשר מחשב אחד משדר תו למחשב אחר, הסיביות המשודרות מאוחסנות בחוצץ שבכרטיס התקשורת של המחשב הקולט. לאחר שכל הסיביות מגיעות, כרטיס התקשורת משחרר את התו מתוך המסגרת ובודק את תקינותו בעזרת סיביות הבדיקה. בסוף תהליך זה, תוכנית התקשורת צריכה לקלוט את התו וכך לפנות את החוצץ, כדי שיוכל לקלוט את המסגרת הבאה.

כדי לדעת אם החוצץ מכיל תו המוכן לקליטה, נפעיל את פסיקת בדיקת הסטטוס ונבדוק את ערך הסיבית הראשונה באוגר ah.



כתבו פונקציה בשם NewCharReceived, המחזירה "אמת" במידה והתקבל תו חדש במלואו.

כדי לקלוט את התו מתוך החוצץ של כרטיס התקשורת, נפעיל את שירות מספר 2 של פסיקת התקשורת הטורית (14H).

לפני הפעלת הפסיקה:

אוגר ah - צריך להכיל 2 (מספר השירות).

ואוגר dx - צריך להכיל את מספר החיבור (0 עבור COM1, או 1 עבור COM2).

לאחר הפעלת הפסיקה :

אוגר al - יכיל את התו שהתקבל.



כתבו פונקציה בשם GetChar הקולטת תו מתוך החוצץ שבכרטיס התקשורת.

17.6 יצירת תוכנית צ'אט בין שני מחשבים

עכשיו כשאנו יודעים כיצד לשלוח ולקלוט תווים, נוכל להכין בקלות רבה תוכנית צ'אט (שיחה, או סתם פטפוט) בין שני מחשבים. הצ'אט צריך להיות דו-כיווני, שבו שתי התחנות יכולות לשדר תווים. כל תו שיוקש במקלדת של תחנה אחת, ייכתב על המסך של התחנה השנייה, ולהפך.

מבנה התוכנית הבסיסי ביותר צריך להיראות כך :

כל עוד לא נלחץ מקש ESC,

המתינו להקשת תו במקלדת, או לקבלת תו מהתחנה השנייה.

אם נלחץ תו במקלדת, שדרו את התו.

אחרת, קלטו את התו שהתקבל והציגו על המסך.

התוכנית הבאה מדגימה צ'אט בסיסי בין שני מחשבים.

הריצו את התוכנית וראו כיצד היא פועלת. כדי להפעיל את התוכנית, עליכם לחבר את שני המחשבים בעזרת **כבל טורי** - **Serial Link**, הנקרא גם **כבל מוצלב** או **Null-Modem**.

```
/*** Com.C ***/

#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define ACK 0x06
#define NAK 0x15

typedef unsigned char byte;

int InitCom (int Com, int Speed)
/*--------------------------------------------------------------------------
      פונקציה לאתחול תקשורת טורית בין מחשבים עם בדיקת זוגיות ושתי
      סיביות עצירה, במהירות שיבחר המשתמש.
      פרמטרים מועברים :
         Com  - מספר היציאה.
         Speed - המהירות בסל"ש.
      ערך מוחזר :
         "אמת" בהצלחה ו"שקר" בכישלון.
 --------------------------------------------------------------------------*/
{
   union REGS ireg;
   byte Properties;               // בית מאפייני השידור

   switch (Speed)                 // קביעת מהירות החיבור
   {
      case 110 : Properties = 0;       break;
      case 150 : Properties = 1;       break;
      case 300 : Properties = 2;       break;
      case 600 : Properties = 3;       break;
      case 1200: Properties = 4;       break;
      case 2400: Properties = 5;       break;
      case 4800: Properties = 6;       break;
      case 9600: Properties = 7;       break;
      default  : return (0);
   }
   Properties = Properties << 5;  // הזזת ערך המהירות שיתאים למבנה המאפיינים
   Properties += 0x1f;            // בדיקת זוגיות זוגית ושתי סיביות עצירה
   if ( Com==1 || Com==2 )
      ireg.x.dx = (--Com);        // הגדרת היציאה
```

```
   else
      return (0);
   ireg.h.ah = 0;                 // אתחול התקשורת - 0
   ireg.h.al = Properties;
   int86 (0x14,&ireg,&ireg);      // הפעלת פסיקת התקשורת
   ireg.h.ah = 3;                 // בדיקת סטטוס - 3
   int86 (0x14,&ireg,&ireg);      // הפעלת פסיקת התקשורת
   if (ireg.h.ah & 0x8)           // אם התרחשה תקלה
      return (0);
   else
      return (1);
}

int SendChar (int Com, char Ch)
/*---------------------------------------------------------------------------
                                                    פונקציה לשידור תו.
                                                    פרמטרים מועברים :
                                                    Com  - מספר היציאה.
                                                    Ch - התו לשידור.
                                                    ערך מוחזר :
                                                    "אמת" בהצלחה ו"שקר" בכישלון.
 ---------------------------------------------------------------------------*/
{
   union REGS ireg;

   if ( Com==1 || Com==2 )
      ireg.x.dx = (--Com);           // הגדרת היציאה
   else
      return (0);
   do{
      ireg.h.ah = 3;                 // בדיקת סטטוס - 3
      int86 (0x14,&ireg,&ireg);      // הפעלת פסיקת התקשורת
   } while   (!(ireg.h.ah & 0x20));  // כל עוד התחנה משדרת

   ireg.h.ah = 1;                    // שליחת בית - 1
   ireg.h.al = Ch;                   // התו לשליחה

   int86 (0x14,&ireg,&ireg);         // הפעלת פסיקת התקשורת
   return(1);
}
```

```
int GetChar (int Com, char *Ch)
/*---------------------------------------------------------------------------
        פונקציה לקבלת תו.
        פרמטרים מועברים :
          Com  -  מספר היציאה.
          *Ch - מצביע לתו שהתקבל.
        ערך מוחזר :
         "אמת" בהצלחה ו"שקר" בכישלון.
 ---------------------------------------------------------------------------*/
{
   union REGS ireg;
   if ( Com==1 || Com==2 )
      ireg.x.dx = (--Com);        // הגדרת היציאה
   else
      return (0);
   ireg.h.ah = 2;                 // קבלת בית - 2
   int86 (0x14,&ireg,&ireg);      // הפעלת פסיקת התקשורת
   *Ch = ireg.h.al;               // התו שהתקבל
   return(1);
}

int NewCharReceived (int Com)
/*---------------------------------------------------------------------------
        פונקציה הבודקת האם התקבל תו חדש.
        פרמטרים מועברים :
          Com  -  מספר היציאה.
        ערך מוחזר :
         "אמת" אם התקבל תו חדש ו"שקר" אם לא.
 ---------------------------------------------------------------------------*/
{
   union REGS ireg;

   if ( Com==1 || Com==2 )
      ireg.x.dx = (--Com);        // הגדרת היציאה
   else
      return (0);
   ireg.h.ah = 3;                 // בדיקת סטטוס - 3
   int86 (0x14,&ireg,&ireg);      // הפעלת פסיקת התקשורת
   return (ireg.h.ah & 1);        // החזר אמת אם התקבל בית חדש
}
```

```
void main (void)
{
   char ch;
   int com;
   printf("Enter Com Number");
   scanf("%d",&com);
   if (!InitCom(com,9600))        // אתחול התקשורת
   {
      printf("Init COM failed");
      exit(1);
   }
   clrscr();
   while (ch != 27)               // כל עוד לא נלחץ ESC
   {
      while ( !kbhit() && !NewCharReceived(com) );
                                  // לולאת המתנה ללחיצת מקש או לקבלת בית חדש
      if (kbhit())                // אם נלחץ מקש
         {
            ch = getch();
            SendChar(com,ch);     // שדר את התו שנקלט
         }
      else                        // אם התקבל תו חדש
         {
            GetChar(com,&ch);     // קלוט את התו שהתקבל
            if (ch != ACK )
               putch(ch);         // הצג את התו שהתקבל על המסך
            if (ch == 13)         // אם התקבל ENTER
               printf("\n");
            SendChar(com,ACK);
         }
   }
}
```

אם קראתם בעיון את התוכנית, ודאי שמתם לב לכך, שבכל פעם שהתקבל תו, שלחנו את הקבוע **ACK**. הקבוע ACK (Positive acknowledge), הוא קבוע מוסכם הנשלח כאישור לקבלת מסגרת תקינה, ואילו הקבוע **NAK** (Negative acknowledge), הוא קבוע מוסכם הנשלח כאשר התקבלה מסגרת לא תקינה (שהתגלתה בה שגיאה).

בתוכנית הכתובה למעלה, שלחנו אישור (ACK) בלי להתחשב בתקינות המסגרת שהתקבלה. כמו כן, לא התייחסנו לקבלת הודעות אישור (ACK) או שגיאה (NAK).



חשבו כיצד נוכל לבדוק את תקינות המסגרת שהגיעה למחשב שלנו, ולשדר הודעה בהתאם.

רמז: חזרו וקראו מהו הערך המוחזר באוגר ah.



נסו לשפר את התוכנית, כך שתטפל בשליחת הודעות על תקינות המסגרת שהתקבלה, וגם תדאג לטפל בקבלת הודעות תקינות על המסגרות ששידרה (אם מתקבלת הודעת NAK - היא תשלח שוב את אותה מסגרת). תוכלו גם לעצב את המסך, כך שיחולק לשני חלקים: החלק העליון יכיל את התווים שנשלחו על ידי המחשב המשדר, והחלק התחתון יכיל את ההודעות שהתקבלו על ידי המחשב הקולט.

כיצד ניצור תוכנית להעברת קבצים בין מחשבים?

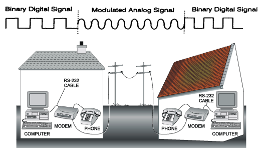
כדי להעביר קובץ שלם בין מחשבים (במקום תווים בודדים), עלינו לפתוח את הקובץ לקריאה (בתחנה המשדרת) ולפתוח קובץ חדש לכתיבה בתחנה הקולטת. במקום להעביר תווים (שגודלם שמונה סיביות), עלינו לקרוא בכל פעם בית (8 סיביות) מהקובץ המקורי, לשדר אותו לתחנה הקולטת (כאילו היה תו רגיל), ושם נכתוב אותו בקובץ החדש.

17.7 המודם

נכון להיום, קווי הטלפון משמשים להעברת אותות אנלוגיים (גלים אלקטרומגנטים שעוצמתם משתנה באופן רציף), ואילו הנתונים במחשב מועברים באמצעות אותות דיגיטליים בינאריים (גלים אלקטרומגנטיים שעוצמתם משתנה באופן בדיד בין אפס לאחד). לכן, כדי לקשר בין המחשב לקו הטלפון, דרוש מכשיר שימיר אותות אנלוגיים לאותות בינאריים ולהיפך. מכשיר זה הוא **מודם** (**Modem**), ששמו מורכב מראשי התיבות של המילים **Mo**dulator-**dem**odulator (המתייחס לפעולות אפנון ומיצוי).

תהליך שידור הנתונים מתבצע כך: הנתונים הדיגיטליים (בינאריים), מועברים מזיכרון המחשב אל המודם בתקשורת טורית. המודם ממיר אותם לאותות אנלוגיים, ושולח אותם דרך קו הטלפון למודם שבמחשב הקולט. המודם במחשב הקולט ממיר את האותות האנלוגיים שמגיעים אליו בקו התקשורת לנתונים דיגיטליים, ומעביר אותם לזיכרון המחשב בתקשורת טורית.

השליטה במודם נעשית על ידי אוסף פקודות הנקראות **פקודות AT** (AT Commands), אשר נקראות כך מכיון שכולן מתחילות בצירוף התווים AT). פקודות אלו פעילות רק כאשר תוכנית התקשורת פועלת מול המודם ועדיין לא הוקם ערוץ תקשורת אל המחשב המרוחק. הפקודות מועברות למודם בתקשורת טורית, כאילו היה מחשב שמחובר בכבל כמו בסעיף הקודם. לאחר הקמת הקשר, הנתונים הנשלחים למודם מועברים ישירות אל מחשב היעד.



מבנה הפקודה	משמעות
AT	בדיקת התקשורת עם המודם. אם התקשורת תקינה המודם יענה OK.
ATDP טלפון	חיוג בשיטת חיוג מתקפים למספר "טלפון".
ATDT טלפון	חיוג בשיטת חיוג צלילים למספר "טלפון".
ATDTx טלפון	חיוג הספרה x, המתנה של מספר שניות וחיוג מספר הטלפון (מיועד למשרדים עם מרכזיה, בהם יש להקיש ספרה כלשהי לפני חיוג לקו חוץ).
ATH	ניתוק התקשורת.
ATA	ענה לצלצול (משמש כאשר מזהים צלצול מצד מרוחק ורוצים לענות למודם המרוחק ולהתחבר אליו).
+++	העברת המודם ממצב חיבור לקו למצב בו ניתן לשלוח לו פקודות AT. כדי לחזור לחיבור, נשלח את הפקודה ATA. כדי לנתק את התקשורת, נשלח את הפקודה ATH.

הערה

בסוף כל פקודה יש לשלוח תו ENTER (13 בקוד ASCII).

בתגובה לפקודות AT המודם משדר למחשב את אחת התשובות האלו:

התשובה	משמעות
OK	הפקודה התבצעה בהצלחה.
ERROR	הפקודה היתה שגויה ולא התבצעה.
NO CARRIER	בצד השני ענו לצלצול אך עדיין לא בוצע החיבור למודם (אין זיהוי של גל נושא).
קצב שידור CONNECT	החיבור בוצע בהצלחה על פי "קצב שידור" (קצב השידור נקבע על פי יכולת השידור של המודם האיטי בין השניים).
NO ANSWER	אין תשובה מהצד המרוחק (המודם המרוחק לא ענה לצלצול).
NO DIAL TONE	אין צליל חיוג ולא ניתן לחייג.
BUSY	הקו תפוס.
RING	התקבל צלצול מצד מרוחק.

17.8 יצירת חייגן טלפון

הדרך הפשוטה ביותר ליצור חייגן טלפון, היא להתייחס ליציאת התקשורת הטורית של המודם, כאילו היתה קובץ (כפי שעשינו בפרק 5 כששלחנו נתונים למדפסת). אפשרות אחת היא לעשות זאת בעזרת stdaux (ראה פרק 5 סעיף 5.4). הבעיה היא שאין לנו שליטה על היציאה ש-stdaux מצביע אליה. לכן אנו עלולים, לדוגמה, לשלוח נתונים לעכבר במקום למודם. לכן, עדיף שניצור את הקובץ המדומה בעצמנו.

כל מה שעלינו לעשות הוא, להגדיר מצביע מסוג מצביע לקובץ, ואחר כך לפתוח קובץ בשם ?COM (את סימן השאלה נחליף במספר היציאה שהמודם מחובר אליה. שימו לב שיש להשתמש באותיות גדולות).

לדוגמה, פתיחת קובץ מדומה ליציאה com1:

```
FILE *Modem = fopen ("COM2","w+t");
```

לאחר הגדרת היציאה נוכל לשלוח נתונים למודם, כאילו היה קובץ. זכרו שהתוכנית לא תפתח קובץ בשם COM1, אלא תתייחס ליציאה הטורית COM1 כאילו היתה קובץ.

כדי ליצור תוכנית חייגן, כל שעלינו לעשות הוא לכתוב לקובץ המדומה את הפקודה ATDT בצירוף מספר הטלפון הרצוי. כדי לסיים את השיחה, נכתוב לקובץ המדומה את הפקודה ATH.

```
/*** CALL.C ***/

#include <stdio.h>
#include <conio.h>

void main (void)
{
   FILE *Modem = fopen ("COM2","w+t");
   char Phone[13], Command[20];
   gets(Phone);
   sprintf(Command,"ATDT%s\n",Phone);
   fprintf(Modem,"%s",Command);
   getch();
    fprintf(Modem,"ATH\n");
   fclose(Modem);
}
```

נסו לעצב תוכנת חייגן בעלת ממשק, הנראה כמו טלפון עם לחצנים. הוסיפו לחייגן גם ספר טלפונים, שבו ניתן יהיה לשמור מספרי טלפון רצויים, ולצלצל מתוכו. אם תרצו, תוכלו אף לשלב את קבצי הקול, המצורפים לתיקיה של פרק זה (תחת תיקיה books\59281), המכילים צלילים שונים שמפיק טלפון. כך, כל הקשה על ספרה במסך, תגרום להשמעת הצליל המתאים לה. הרי לכם פרויקט תרגול מושלם, פשוט ונחמד.

17.9 שידור וקליטת תווים באמצעות המודם

השיטה לכתיבה אל יציאת המודם כאילו היה קובץ, אמנם קלה ופשוטה, אך היא אינה מאפשרת לקבל את התשובות שהמודם עונה לפקודות ה-AT (לפני הקמת הקשר), או את הנתונים שהמודם קולט (אחרי הקמת הקשר). לכן, אם ברצונכם לכתוב תוכנית שתשתמש בשאר היכולות של המודם (חוץ מפעולת החיוג), עליכם לתקשר עם המודם בדיוק כפי שתקשרנו עם המחשב המחובר בכבל טורי. נסו להפעיל את תוכנת הצ׳אט שכתבתם, ובחרו ביציאה שהמודם מחובר אליה, כתבו אל המודם פקודות AT, וראו כיצד הוא עונה לכם.

כך תוכלו ליצור תוכנת צ׳אט בין מחשבים, באמצעות המודם או תוכנה להעברת קבצים דרך המודם.

בסעיף 17.3 (בדיקת סטטוס התקשורת), הזכרנו כי בנוסף למידע המוחזר באוגר ah, שירות בדיקת הסטטוס, מחזיר באוגר al מידע המתייחס למודם, ואשר עשוי לשמש אתכם בתוכנית.

המידע נשמר באוגר al באופן הבא, ומשמעותו מוצגת כאשר הסיבית המתאימה מורמת:

b3-b0 – סיביות חסרות משמעות.

b4 – כרטיס התקשורת קיבל אישור CTS לשידור תו (Clear To Send).

b5 – כרטיס התקשורת קיבל אישור DSR. המודם או המחשב בצד המרוחק פעילים ומחוברים לכרטיס התקשורת (Data Set Ready).

b6 – התקבל צלצול מקו התקשורת.

b7 – מתקיימת תקשורת בין המודמים. ברגע הניתוק, הסיבית תרד ל-0.

הערה

המידע בפרק זה, מתבסס בעיקר על הספר "תקשורת נתונים" של האוניברסיטה הפתוחה. למידע נוסף בכל הקשור לתקשורת נתונים מומלץ לעיין בספר זה, או בספרי הוצאת הוד-עמי המפורטים בסוף הספר.

נספח א׳

מבט מעבר לחומר השגרתי

נכתב על ידי אסף שלי

הלימוד בפרק זה יאפשר לאוהבי מחשבים מבט אחד קדימה, אל מעבר לחומר השגרתי. "מבט קדימה", מכיון שבפרק זה לא נלמד חומר תכנותי עיוני, אלא חומר מעשי ואפילו פיקנטי. קשה מאוד ללמוד חומר כזה מקריאה, אבל שעות רבות של חוסר שינה נחסכות, אם יש לאן לפנות, כאשר נתקלים בבעיית תכנות.

בפרק זה תמצאו רשימה של מילים שמורות בשפת C, אשר לא מקובל להזכירן במסגרת הלימוד הסדיר, רשימת טריקים מיוחדים שתדריך אתכם להימנע משגיאות בזמן ריצה או משגיאות תחביר, ומספר טכניקות תכנות מתקדמות, אשר משמשות כלי בעל עוצמה בידי מתכנת שמכיר אותן היטב.

אין לי כל כוונה ללמד חומר תכנות, ועל הנושאים שכלולים בפרק זה לא ניתן לתת שיעורי בית, או למדוד את מידת ההצלחה בעזרת ציון (כמה חבל...). למעשה, השתדלתי לרכז בפרק זה את מירב החומר שלא נסקר במקום אחר, ובכלל זה, גם מספר טיפים ועצות שנובעים מנסיוני בפיתוח מערכות מבוססות C.

ראוי לציין, כי חלק משמעותי מהחומר שמופיע בפרק זה אינו מתאים ללימוד בבתי ספר, וחלק מהטריקים שמוסברים אפילו אסורים לשימוש, בזמן שאחרים מומלצים בהחלט. מטרת הפרק להכשיר מתכנת ממוצע לעבודה רצינית בשפת C. לשם כך, עליו להכיר ולדעת נושאים מורכבים רבים, שאינם במסלול הלימוד הרגיל. נושאים אלה מובאים ברובם על קצה המזלג, ואת הלימוד האמיתי ניתן להשיג רק על ידי התנסות בתכנות ממש.

הערה

ספר זה דן כולו בשפת C, לכן השתדלתי מאוד להימנע מלהיכנס להסברים על שפת ++C, אם כי רבים מהדברים מתאימים גם לשפה זו.

עם זאת יש לציין, כי רוב החומר אינו מוגדר בהכרח כ- ANSI C תקני, אלא כולל הרחבות תואמות ל- Turbo C3, וחלקן המכריע תואמות מהדרים מתקדמים יותר, דוגמת Borland C 5.02, C Builder או Visual C++.
במקרים של הבדלים משמעותיים ציינתי זאת כהערה.

פנו לקטלוג שבסוף הספר ובתקליטור כדי לקבל מידע על ספרים נוספים.

אודות שפת C

כאשר לוחצים בעכבר על תפריט HELP ועל ABOUT, נהוג לקבל פרטים על יצרן התוכנה ועל הגירסה. אילו יכולנו לבדוק Help > About על שפת C עצמה, זה מה שהיה כתוב :

שפת C

נכתבה על ידי דניס ריצ'י

(Dennis M. Ritchie)

לסיוע בכתיבת מערכת ההפעלה Unix

במעבדות בל.

שפת C היא שיפור של שפת B, שנועדה לשימוש פנימי של מעבדות בל (שפת B – על שם Bell). השפה פותחה בתחילת שנות השבעים של המאה שעברה (כלומר המאה ה-20). בשנת 1983 הוגדר תקן ANSI של השפה ומשלב זה החלה החדירה שלה לשוק. שפת C פותחה כדי לאפשר כתיבה נוחה של מערכת ההפעלה יוניקס (Unix). ראוי לציין שמהדר לשפת C מובנה כחלק ממערכת ההפעלה יוניקס על גרסאותיה השונות.

הפקודות שכלולות בקבצים DOS.H ו- WINDOWS.H, הן הרחבות מיוחדות למחשבי PC (תואמי IBM). לעומת זאת, קיימות מספר פקודות למערכת Unix שאינן נתמכות בהכרח על ידי מחשבי PC, כמו למשל, הפקודה fork שמאפשרת לתוכנית לרוץ במקביל לעצמה. פקודה כזו אינה אפשרית ב-DOS, אולם היא נתמכת על ידי Windows באופנים שונים, שביניהם הקרובה ביותר למקור היא הגירסה של Windows NT.

תכונות שפת C

אחת התכונות העיקריות ביותר של שפת C, היא הפשטות שבה. אין הכוונה לפשטות עבור המתכנת (דוגמת Java או Visual Basic), אלא דווקא פשטות עבור מי שבונה את המהדר.

שפת C "אוהבת" מאוד לתרגם כל פקודה ואופרטור למספר הקטן ביותר של פקודות מכונה (או פקודות אסמבלי).

בשפת C יש למתכנת שליטה מקסימלית על המחשב, ורק האוגרים של המעבד והטיפול במחסנית נמצאים בידי השפה.

בשפת C ניתן לבצע כל פקודה שהמעבד 8086 (תואם IBM) מסוגל לבצע, מלבד הפקודה Rotate, אשר השימוש בה מיועד לעבודות המפעילות גרפיקה מתקדמת בלבד. יש פונקציות ספריה שתומכות בפקודה זו, ולא קשה ליישם אותה בעזרת הזזה.

קריאות לפקודות משתמשות במוסכמות של מערכת ההפעלה ושל BIOS. לכן, קל מאוד להתאים דרייבר שנכתב בשפת C לעבודה על מחשב PC, ובקלות יחסית ניתן לכתוב בשפת C תוכנית שוכנת זיכרון – TSR (Terminate Stay Resident).

פונקציות ספריה רבות פונות אל מערכת ההפעלה – דוגמת טיפול בקבצים, מקלדת, מסך ועכבר. הן אמורות רק לסדר יפה את הפרמטרים באוגרים, לבצע את הפסיקה, ולהחזיר את הערך המוחזר כפי שהוא, ולמעשה – לא לגעת בערך המוחזר, כדי שיועבר הלאה. זהו יתרון עצום של השפה. תוכניות שכתובות בעזרת פקודות אלו תהיינה מהירות בהרבה מתוכניות שמבצעות מספר בדיקות והמרות לפני ביצוע הפסיקה עצמה ואחריה. למעשה, בשפת C לכל פסיקה של מערכת ההפעלה ניתן שם, והפרמטרים סודרו לביצוע קריאה מסודרת.

תכונה נוספת וחשובה של השפה היא, שכל משפט שהמתכנת כותב נחשב לביטוי מתמטי: כל פקודה, כל פעולה של אופרטור ואפילו כל פונקציה. בשפת C אין פרוצדורות, ובמקומן פונקציות המחזירות ערך "כלום" – void. למעשה, תקן ANSI C מתיר שימוש בפונקציות בלבד, ורק הפונקציה הראשית ()main אינה חייבת להחזיר ערך.

כדוגמה לפקודה חוקית, נבחן את ה"ביטוי" המתמטי הבא:

```
1;
```

הערה "1" מחושב לביטוי מתמטי, כפקודה בתוך פונקציה, ולא כהצהרה על משתנה גלובלי.

מכיון שלפנינו ביטוי מתמטי, המהדר יקבל אותו כחוקי.

תכונה נוספת היא מצביעים. למה להסתבך עם מחרוזות ומבנים שונים ומשונים, ששפות עיליות אחרות משתמשות בהם, אם אפשר להשתמש באותה השיטה שבה משתמשת "תוכנת המערכת עצמה".

שפת C משתמשת במצביעים כי כך בנויה המערכת, ואין צורך לבצע המרות שגוזלות זמן בפונקציות הספריה.

כבונוס מיוחד, המתכנת מקבל שליטה מלאה על המחשב ועל הזיכרון שבו. כמו כל כלי חזק אחר, במצביעים צריך לדעת להשתמש כראוי. המתכנת מקבל את מלוא הכלים ומלוא הגישה אל מערכת ההפעלה ואל החומרה במחשב. מתכנת שמפחד מכך, אולי יעדיף שפות עיליות יותר, דוגמת אלו שמבוססות על בייסיק או על פסקל.

הערה **פונקציות ספריה - פונקציות שהם חלק משפת C ומוגדרות בקבצי הכותר (header) - h.*.**

אופרטורים מיוחדים

סימני דרך רבים טבועים בשפת C, סימנים שמעידים כי מי שפיתח אותה אהב מאוד תכנות בשפת אסמבלי. מבין הדברים הבולטים נזכיר את השימוש במצביעים ואת השימוש באופרטורים =% ,=/ ,=* ,=- ,=+. אופרטורים אלה משפרים את ביצועי התוכנית והם גם מאוד הגיוניים למתכנתי אסמבלי. לדוגמה, כדי לחבר שני ערכים זה לזה, יש להציב ערך אחד במשתנה ואז להוסיף לו את הערך השני. לכן, במקום לכתוב A=A+1 נכתוב A+=1 (בדומה לפקודה ADD ax,1 באסמבלי). מהדר ללא אופטימיזציה (תכונה שקיימת במהדרים

החדשים ומאפשרת להם למצוא את הדרך היעילה ביותר לבצע פקודות), ייבצע שתי פעולות לפתרון הביטוי A=A+1: תחילה, הוא מחשב את הביטוי החלקי (A+1), לאחר מכן, מציב ב- A. הביטוי A+=1 עם האופרטור =+ עושה זאת כפקודה אחת.

אם עדיין לא שמנו לב, האופרטור צריך להיות =+ בשל הכתיבה A=A+1, ולא =+ בדומה ל- A=+A1. סדר האופרטורים במקור היה כזה (=+). הבעיה היא שהיה צורך להרחיק ערכים שליליים מסימן השוויון, אחרת לא ניתן להבדיל בין שני אופרטורים '-' ו- '=' צמודים לבין אופרטור יחיד '=-', כך: A=-1, כיון שאם לא היינו משאירים רווח, היינו מקבלים A=-1 שבמקור היה A=A-1, ואחר כך היינו יושבים שעות לדבג, כדי למצוא את הרווח האבוד. בשל כך יצאה גירסה מחודשת של השפה (עוד בחיתוליה), שמתקנת את השגיאה.



דיבוג - ניפוי שגיאות (Debugging), על שם התקלה הראשונה במחשבים בעולם, חיפושית (Bug) שקיצרה מעגלים חשמליים (לך תמצא חיפושית בערמת ג'וקים חשמליים בגודל של בניין). Debug - ההיפך מ- Bug, או הוצאת ה- Bug.

קבוצת אופרטורים נוספת היא '++' (פלוס פלוס), ו- '--' (מינוס מינוס). בשפת אסמבלי קיימות הפקודות INC ו- DEC , שפת C יורשת לכן את האופרטורים ++ עבור INC ו- -- עבור DEC. מהדר ללא אופטימיזציה טובה ייצור מהביטוי A+=1 את הפקודה ADD A,1, וייצור מהפקודה A++ את הפקודה INC שהיא מהירה בהרבה. דבר זה משמעותי מאוד בלולאות מהירות, כמו לדוגמה, צביעת כל המסך בירוק.

אופרטור מאוד לא אהוב, שגם הוא נובע משפת אסמבלי, הוא האופרטור הטרינרי (TRI=שלוש). קיים אופרטור טרינרי אחד בלבד בשפת C, אשר מבצע פעולה דומה לזו של משפט if-else. בשפת אסמבלי מבצעים השוואה, ואם התנאי מתקיים, קופצים לכתובת מסוימת כדי לבצע משהו, אחרת ממשיכים באותה הנקודה. תוכניתן אסמבלי, רגיל לפקודה בודדת, שמשמעותה "אם 'כן' – אז ככה, ואם 'לא' – אז אחרת". כך למשל, אם A גדול מ-5, אז B=1, אחרת B=0:

```
if (a>5) b=1; else b=0;
```

משפט תנאי זה יתורגם ל:

```
(a>5)?(b=1):(b=0);
```

אין צורך לכתוב את סימני הסוגריים, אך כך נוח יותר להפריד בין הביטויים.

שפה מתמטית

כבר הזכרתי ששפת C היא שפה מתמטית מאוד, אבל יש בה יותר מהנגלה לעין.

סימן/אופרטור פסיק

לסימן ה"פסיק" (,) בשפת C יש שימושים רבים, שחלקם אינם ידועים למתכנתים רבים.

השימוש הראשון, הידוע לכולם, הוא השימוש בפסיק כ**אופרטור** (operator), המפריד בין פרמטרים בהצהרה של פונקציה או בין המשתנים בהצהרת המשתנים.

אולם לפסיק יש שימוש נוסף, הוא משמש גם כסימן **פיסוק** (punctuator). מקורו של שימוש זה הוא בשפת B (השפה שקדמה לשפת C, ושפת C מבוססת עליה). בשפת B שימש הסימן פסיק כמפריד בין שתי פקודות. בשפת C הוחלט להוסיף את סימן ה"נקודה-פסיק" (;) שמשמעותו היא הפרדה מוחלטת בין שתי פקודות, ואילו הסימן "פסיק" הפך להפרדה חלקית בין פקודות, תוך שמירה על המשמעות הלוגית.

זה נשמע יותר מסובך ממה שזה, ולכן ניקח דוגמה מוכרת של לולאת for:

```
int strncmp (char *s1, char *s2, int n)
{
   int i;
   for ( i = 0; (*s1==*s2) && (i<n) ; i++ , s1++,s2++);
   return (i==n);
}
```

בדוגמה זו, סימני הנקודה-פסיק (;) מהווים את החלוקה הלוגית הרגילה של לולאת for:

```
for (אתחול;בדיקה;קידום);
```

ואילו סימני הפסיק (,) משמשים לחלוקה פנימית, וכך מתאפשר לקדם גם את i, גם את s1 וגם את s2. הסימן נקודה-פסיק, יותר "חזק" מהסימן פסיק, בדיוק כפי שנהוג בכללי הניקוד.

אם כך, נסו לחשוב מה משמעות הפקודה הבאה:

```
Arr[3,4] = 0;
```

הפקודה מציבה לתוך המערך החד-מימדי Arr בתא החמישי (מספר 4) את הערך אפס!

אני בטוח שרבים מכם (במיוחד אלו המכירים שפות תכנות אחרות כמו פסקל, למשל), טעו בעבר וניסו לפנות למטריצה בעזרת זוג אחד של סוגריים מרובעים, ולא הבינו מדוע זה לא עובד כראוי.

בשפת C, כדי לפנות לשורה מספר 3 בטור מספר 4, יש להשתמש בשני זוגות של סוגריים מרובעים:

```
Mat[3][4] = 0;
```

כאשר כתבנו [3,4], המהדר התייחס לזה כמשמעות לוגית אחת המציינת את מספר התא. בתחילת נספח זה למדנו שהפקודה

```
1;
```

היא פקודה חוקית (חסרת משמעות אבל חוקית), ולכן גם הפקודה:

```
3,
```

היא פקודה חוקית. ולכן המהדר יתייחס לפקודה 3 כפקודה, יבצע אותה, ואחר כך ידלג אחרי הפסיק ויציב בתא מספר 4 את הערך אפס.

אם הבנתם את הדוגמה הזו, נסו להבין מה תבצע הפקודה הבאה:

```
x = (1,2);
```

הפקודה תציב במשתנה x את הערך 2 !

נשמע מטופש!! אז מה תבצע הפקודה הבאה:

```
a = (( fptr=fopen("file.txt","w+b")),getc(fptr));
```

הפקודה תפתח את הקובץ ותקרא ממנו תו אחד לתוך המשתנה a.

הערה

זכרו שהקוד שאתם כותבים צריך להיות קריא ככל האפשר, כך שכל מתכנת אחר שיקרא אותו יבין אותו בקלות.

הגדלה/הקטנה מוקדמת/מאוחרת

למען האמת, יכולתי להשתמש בטריק מתמטי יותר מוכר: "הגדלה מוקדמת". האופרטור '++' והאופרטור '- -' יכולים להופיע לפני, או אחרי המשתנה שעליו הם פועלים. כאשר האופרטור '++' מופיע לפני המשתנה, ערכו של המשתנה גדל ב- 1, והביטוי מחושב לערכו לאחר ההגדלה, כמו למשל:

```
A=5;
B=++A;
```

ערכו של B הוא 6, וגם ערכו של A הוא: 6.

דוגמה אחרת:

```
A=5;
B=A++;
```

בסיום הפעולה ערכו של A יהיה 6, כי הגדלנו אותו, אבל B קיבל את ערכו של A לפני ההגדלה, לכן B יהיה שווה ל-5.

הגדלה מוקדמת קרויה באנגלית **Pre Increment**,
והגדלה מאוחרת קרויה **Post Increment**.

הצבה בתוך ביטוי הוא טריק שימושי נוסף של שפת C, ואילו שפות אחרות מנסות להימנע ממנו:

```
B=a=5;
B=(a=5);
B=6+(a=5)+3;
```

בשני המקרים הראשונים ערכו של B הוא: 5, במקרה האחרון ערכו הוא: 6+5+3=14, ובכל המקרים הצבנו ב- a את הערך 5.

שימוש נחמד לאפשרות זו, נמצא בפקודות אלו:

```
FILE* fp;
if (fp=fopen("file.txt","w+b")) fclose(fp);
 else puts("HA?! No such file.\n");
```

קיימות פקודות רבות, שרצוי לוודא כי התבצעו כראוי, לדוגמה, פקודות הפועלות על קבצים, או פקודות להקצאת זיכרון. כדי לעבוד באופן תקין, עלינו לבדוק את הערך המוחזר מייד לאחר ביצוע הפקודה ולפני שנשתמש בו. חשוב במיוחד להקפיד על נוהל זה בעת פעולה על מצביע.

```
FILE *fp=0;
if (!(fp=fopen("file.txt","w+b")))
 { puts("No file to read"); exit(0); }
fsacnf(fp,"%s",char_ptr1);
fclose(fp);
```

שפת C מתירה להשתמש בהצבה בתוך משפט תנאי, אך יש בכך קושי מסוים. אופרטור ההשוואה זהה לאופרטור ההצבה במראה ובמיקום על המקלדת. האחד כפול מהאחר, אופרטור ההשוואה הוא'==' ואופרטור ההצבה הוא'='. לעיתים, מתכנתים אינם לוחצים פעמיים על המקש ואינם מקבלים את האופרטור הרצוי. כתוצאה מכך, התוכנית לא תפעל כראוי. דוגמה לתקלה כזו: הביטוי (A=5) תמיד שונה מאפס, לכן ערכו תמיד יהיה "אמת" ואין כל משמעות לפקודה if:

```
if (a=5) return(1);
```

בנקודה זו הפונקציה תסתיים תמיד. המהדר יתריע על הצבה במשפט תנאי, ועל "קוד מת" שאינו מתבצע לעולם. המהדר גם יכול להודיע על ביטוי שמחושב תמיד לאותו הערך ("expression is always true").

אפשר לפתור בעיה זו בקלות: במקום לרשום (A==5), נרשום (5==A), כי מותר להשוות את הערך 5 לערך של A, אך אסור לעומת זאת להציב את A ב- 5. אם בטעות שכחנו סימן '=' אחד, המהדר יודיע על שגיאת תחביר ויעצור.

```
if (5==a) return(1); // OK.
if (5=a) return(1);  // syntax error
```

הערה

מתכנת מתחיל, משקיע מינימום זמן בכתיבת הקוד. מתכנת מנוסה, ישקיע יותר בזמן הכתיבה ויחסוך בכך הרבה יותר מזמן הדיבוג. מתכנתים מנוסים רבים מקפידים על הקפת כל ביטוי בסוגריים. הקפת ביטויים בסוגריים מונעת שגיאות, שנובעות מקדימות אופרטורים, בעיקר בשפת ++C.

הזכרתי ששפת C לא אוהבת לבצע הרבה פקודות, ובעיקר פקודות מיותרות. נבחן את הביטוי הבא: "אם אני נמצא בישראל ובתל-אביב, אז אני תל-אביבי". אם אני לא נמצא בישראל, נשמע מטופש לבדוק האם אני בתל-אביב. שפת C לא תבדוק. נשמע מובן מאליו... האומנם?

```
if ( (InIsrael()==1) && (InTelAviv()==1) )
      puts("sababi babi.");
```

בתחילה תקרא הפונקציה ()InIsrael, אם הפונקציה תחזיר 1 אז סימן שאני בישראל. לאחר מכן, תקרא הפונקציה ()InTelAviv, ואם הערך החוזר ממנה הוא 1 אז יודפס המשפט. אם הפונקציה הראשונה תחזיר אפס, הרי שאין טעם לקרוא לבדיקה השנייה, ואכן הפונקציה השנייה בכלל לא תקרא.

דוגמה נוספת, מורכבת יותר: "אם הצלחתי לפתוח שני קבצים ולהשוות ביניהם, אז אני רושם הודעה למשתמש וסוגר אותם".

```
FILE *fp1, *fp2;
if ( (fp1=fopen("system.dat","rb")) &&
     (fp2=fopen("system.1st","rb")) && CheckFiles(fp1,fp2)  )
 {
  puts("Yes OK.");
  fclose(fp1);
  fclose(fp2);
 }
```

הסבר: אם הצלחתי לפתוח את קובץ "system.dat" לקריאה, וגם הצלחתי לפתוח את קובץ "system.1st" לקריאה, ופונקציית ההשוואה החזירה 1, אז אני מודיע למשתמש. עד כאן הכל טוב. ברור שצריך לסגור את שני הקבצים גם אם ההשוואה לא הצליחה. כלומר, הקבצים פתוחים, אבל אינם שווים. לכן יש להעביר את סגירת הקובץ לאחר משפט התנאי. בכל מקרה, בלי להתחשב בכך שקבצים שווים או לא, יש לסגור אותם. אבל יש לסגור קבצים רק אם פותחים אותם תחילה. אם הקובץ נפתח, המצביע מכיל ערך, אם הפתיחה נכשלה, המצביע מכיל אפס. לכן:

```
FILE *fp1, *fp2;
if ( (fp1=fopen("system.dat","rb")) &&
     (fp2=fopen("system.1st","rb")) && CheckFiles(fp1,fp2)  )
 {
  puts("Yes OK.");
 }
if (fp1) fclose(fp1);
if (fp2) fclose(fp2);
```

ההנחה היא, שאם פתיחת הקובץ נכשלה, ערך המצביע לקובץ הוא אפס. אחרת ערכו שונה מאפס, ואז יש קובץ פתוח שצריך לסגור. הבעיה היא, שאם פתיחת הקובץ הראשון נכשלה, הבדיקה של התנאים תיפסק. עכשיו נבדוק אם הקבצים פתוחים. הבדיקה של הקובץ הראשון תניב ערך אפס, כי הפתיחה נכשלה. הבדיקה של הקובץ השני תניב ערך לא ידוע, כי לא היה ניסיון לפתוח אותו, והוא מעולם לא אותחל לערך כלשהו.

אפשר לכתוב כך את הקוד:

```
FILE *fp1=NULL, *fp2=NULL;
if ( (fp1=fopen("system.dat","rb")) &&
     (fp2=fopen("system.1st","rb")) && CheckFiles(fp1,fp2) )
 {
  puts("Yes OK.");
 }
if (fp1) { fclose(fp1); fp1=NULL; }
if (fp2) { fclose(fp2); fp2=NULL; }
```

שימוש פשוט ונפוץ יותר ייראה באופן הבא:

```
if ((a!=0)&&(x\a>b)) { puts("x\a > b"); }
```

הקוד בודק, האם החלוקה של x ב- a נותנת ערך הגדול מ- b. במידה ו-a שווה ל- 0, לא תתבצע החלוקה כלל, והתוכנית לא תבצע פעולה לא חוקית.

הערה

אני ממליץ בחום רב לאתחל תמיד את כל המשתנים, גם אם מציבים בהם ערך מייד לאחר מכן, מכיון שעלול להגיע היום שבו נשנה את הקוד, ואז משתנה לא מאותחל ייצור פתאום בעיות לא מובנות, במקום שאינו קשור לוגית למקום בו התבצע השינוי.

בנוסף, אני מציע לאפס תמיד מצביע ששוחרר בעזרת ()free, או קובץ שנסגר. ותמיד לבדוק, שהערך אינו אפס לפני סגירת קובץ, או שחרור זיכרון.

סיכום

האופרטור && מבצע AND לוגי, ובודק את הערך שלפניו. אם התנאי מתקיים, הוא בודק את הערך שאחריו. לדוגמה: נניח שנתון A && B ; תחילה ייבדק A ואם ערכו אינו אפס, ייבדק B. אם התנאי הראשון אינו מתקיים, אין טעם לבדוק את השני, כי לא יכול להיות ששניהם מתקיימים אם הראשון אינו מתקיים, לכן התנאי כולו נכשל.

האופרטור || מבצע OR לוגי, ובודק את התנאי שלפניו. אם התנאי אינו מתקיים, הוא בודק את השני, כי מספיק שאחד מהם יתקיים כדי שהתנאי כולו יתקיים. אם הראשון מתקיים, אין טעם לבדוק את הביטוי השני, כיון שברור שהתנאי כולו מתקיים, שהרי מספיק רק אחד מהתנאים, כדי לקיים את הביטוי כולו. לכן, כאשר מבצעים AND לוגי ולא בודקים את כל התנאים, לא מבצעים את המשפט המותנה בכלל.

כאשר מבצעים OR לוגי ולא בודקים את כל התנאים, מבצעים את המשפט המותנה כולו.

עכשיו נציג דוגמה "משעשעת": הסברתי קודם לכן, שכל ביטוי בשפת C הוא פקודה חוקית, לכן גם חוקי המשפט הבא:

```
FILE* fp=NULL;
((puts("opening file"),(fp=fopen("autorun.inf","w+"))) &&
 ((puts("file opened"),
(EOF==fprintf(fp,"IconFile=%s",MyIconName))) ||
 puts("written OK")), fp?fclose(fp),puts("HA?!"));
```

קטע הקוד שבדוגמה, מציג ניסיון לפתיחת קובץ. אם הקובץ לא נפתח, מסתיים התהליך. אם הקובץ כבר פתוח, תודפס ההודעה "file opened", ויתבצע ניסיון לכתיבה. אם fprintf() החזירה EOF, פירוש הדבר שהיה כישלון בכתיבה, אחרת יודפס "written OK". בכל מקרה, הקובץ ייסגר אם המצביע אינו אפס (דבר שאינו סביר). אם כאן המצביע אפס, אז משהו ממש לא תקין, למרות שאין שום סיבה שבעולם, שבמיקום זה, ערכו של המצביע יהיה 0. הניסיון מלמד שרצוי לבדוק כל פרט, ולצפות לבלתי אפשרי, כי הוא בהחלט עלול לקרות, מסיבה שפשוט לא חשבנו עליה.

הערה

אמנם הטכניקות שנלמדות בסעיף זה חוקיות ותקניות, אך **רצוי להימנע משימוש בביטויים שאינם קריאים במעבר ראשון על הקוד**. במסגרת עבודות בית-הספר, נאסר להשתמש בטכניקות מסורבלות אלו, והן נזכרות כדי לאפשר הבנה מעמיקה יותר של השפה, וכדי להקנות לך כמתכנת, את רוב הכלים של שפת C.

נכתוב שוב את הדוגמה הקודמת:

```
FILE* fp=NULL;
puts("opening file")
if (fp=fopen("autorun.inf","w+"))
 {
  puts("file opened")
  if (EOF==fprintf(fp,"IconFile=%s",MyIconName))
   { ; }
   else
    {
     puts("written OK")
    }
  if (fp) fclose(fp) else puts("HA?!");
 }
```

כאן המקום לציין מהיכן הגיעו שתי מילים מאוד מוכרות:

```
#define EOF (-1)
#define NULL 0
```

רצוי לדעת, ש-EOF ו-NULL אינן מילים שמורות, אלא שני קבועים.

מצביעים

מערכת מחשב מינימלית חייבת לכלול בתוכה מספר מרכיבים חשובים:

מעבד: בלעדיו אין ביצוע חישובים, אין בדיקת תנאים ואין ביצוע פעולות.

זיכרון: בלעדיו המעבד לא יידע אילו פקודות לבצע, לא יזכור תוצאות של חישובים, ולא יזכור מידע שהכניס המשתמש.

קשר חוץ: המחשב חייב קשר עם העולם החיצון, אחרת לא יהיה לו על מה להגיב, ולא יהיה לו כיצד להגיב.

מחשב מקבל קלט, לרוב פסיקה (interrupt), שמודיעה למעבד על הקלט. המעבד מבצע חישובים והחלטות, ושולח מידע להתקנים חיצוניים. המעבד משתמש בזיכרון, כדי לשמור נתונים שהגיעו, לשמור תוצאות ביניים של חישובים, ולשמור תוצאות, כדי לחזור אליהם במועד אחר.

הזיכרון הוא אוסף של תאים, שכל אחד מהם נקרא בית (byte). המעבד הוא היחיד שמחליט איזה נתונים יישמרו באיזה תאים. כדי שהמעבד יפנה אל תא זיכרון בודד, יקרא ממנו או יכתוב אליו, יש לתת לכל תא זיכרון שם. כאשר פונים אל התא, המעבד מציין את שמו של התא עבור שבב הזיכרון. תפקידו של הזיכרון למצוא את התא המסוים, ולאפשר למעבד לגשת אליו. בפועל, 128MB של זיכרון, הם 128 מיליון תאי זיכרון. הדרך היחידה לפנות אליהם היא לא בשם, אלא במספר סידורי. מספרו הסידורי של תא זיכרון, קרוי **כתובת זיכרון**. שפת C, כמו כמעט כל שפה אחרת, מאפשרת למתכנת להעניק שמות לתאי הזיכרון השונים, ומכיוון שלא אכפת למתכנת היכן יושב נתון בזיכרון כל זמן שהוא קיים, המתכנת מכין למהדר רשימה של משתנים, והמהדר מסדר אותם בזיכרון בצורה הנוחה לו (יחד עם ה- Linker).

בשפת C קיימים מספר סוגים (טיפוסים – types) של משתנים, ולהם גדלים שונים בזיכרון. משתנה מסוג char תופס בית אחד, ומשתנה מסוג long תופס ארבעה בתים או יותר. כדי לפנות אל נתון בזיכרון עלינו לדעת עליו שני פרטים: מיקומו בזיכרון וגודלו בבתים, כלומר, כתובתו של הבית הראשון של המשתנה, ומספר הבתים העוקבים שהוא תופס.

כאשר אנו מגדירים משתנה, אנו מצהירים על הסוג שלו. סוג המשתנה קובע, כיצד יתבצעו עליו פעולות מתמטיות וכמובן מה גודלו בזיכרון. למשל: float, unsigned long, signed char.

כאשר אנו מצהירים על משתנה, אין חובה לציין האם הוא signed או unsigned. אם לא תציין זאת, ראוי שתבדוק מהי ברירת המחדל במערכת שבה אתה פועל. במחשב PC מוגדר לפי ANSI C שהוא signed. למעשה, אין חובה לציין את טיפוס המשתנה, וברירת המחדל

כאן היא int. להלן הצהרה של פונקציה, שמקבלת שלושה פרמטרים, שכולם מסוג signed int, ומחזירה פרמטר מאותו הסוג בדיוק.

```
func1(int a, signed b, signed int c);
```

בלי לשים לב, נהוג להשמיט את הטיפוס של הערך המוחזר מהפונקציה ()main.
שתי ההגדרות הבאות שקולות זו לזו:

```
signed int main(void) {   }
main() {   }
```

בזמן ההידור, מוקצות כתובות לכל המשתנים. המתכנת אינו יודע לפני ההידור מה הן הכתובות של המשתנים השונים, אך ניתן לקבל את הכתובת של משתנה, בעזרת האופרטור & (האמפרסנד), כאשר הוא מופיע לפני שם של משתנה. במקום לקבל את הערך שבתא הזיכרון, מתקבלת כתובת התא.

קיים סוג מיוחד של משתנים, ששומרים כתובות זיכרון. משתנים אלה נקראים מצביעים, כיון שהערך שהם מחזיקים, הוא כתובת של משתנה אחר.

הסברתי, שכדי לגשת למשתנה בזיכרון, יש לדעת את כתובתו ואת גודלו, לכן כאשר מצהירים על משתנה מסוג מצביע, יש להצהיר גם על הטיפוס שעליו הוא מצביע. המצביע עצמו תופס תמיד את אותו גודל הזיכרון, גודל הכתובת, וללא קשר, גודל המשתנה שעליו הוא מצביע. אם מצביע ל- char תופס שני תאי זיכרון, אז גם מצביע ל- long תופס שני תאי זיכרון.

הערה

סידור המשתנים בזיכרון אינו חייב להיות כפי שמצוין כאן. מהדר של Visual Studio **למשל, יסדר את המשתנים בגבולות של** sizeof(int)**. מהדר של בורלנד ++C מאפשר למתכנת להגדיר את שיטת האופטימיזציה של הזיכרון. בנוסף, על מערכות מחשב שונות ומערכות הפעלה שונות גודל המשתנים שונה. שימו לב, שלא מדובר בגודל המצביע בזיכרון אלא באיזה כתובת יכול משתנה להתחיל.**

נבחן את ההצהרות הבאות:

```
char ch1, ch2;
int i1, i2;
char *cptr;
int *iptr;
```

המשתנים יוכנסו בדרך כלל לזיכרון על פי סדר הצהרה. אך שים לב שאין זה הכלל.

נראה כעת את מפת הזיכרון עבור הדוגמה (הנחה כי מצביע בגודל שני בתים):

ch1
ch2
i1
i1
i2
i2
cptr
cptr
iptr
iptr

חשוב לי להבהיר, שמערך, הוא בסך הכל מצביע עם שדה של זיכרון, ששמור במיוחד עבורו. למעשה הביטוי arr[0] שקול לביטוי arr*, והביטוי arr[3] שקול לביטוי (arr+3)*. נבחן את ההצהרות הבאות:

```
int a;
char arr[5];
int z;
```

ומפת הזיכרון:

a
a
arr
arr
[arr0]
[arr1]
[arr2]
[arr3]
[arr4]
z
z

מפת הזיכרון כוללת שני בתים של המשתנה הראשון, שני בתים של המצביע של המערך, חמישה בתים של זיכרון ששמור, ושני בתים של המשתנה האחרון. הביטוי arr[5], מתייחס לבית הראשון של המשתנה z.

כאשר אנו מקדמים מצביע ב- 1, אנו מצפים, שהוא יצביע על האיבר הבא. מצביע ל- int, אמור לגדול בקפיצות של שניים, מכיון שגודל int הוא שני בתים. למשל, כאשר נפעיל את האופרטור ++, יגדל מצביע ל- int בשני בתים.

האמת היא, שמשתנה מסוג int אינו תמיד בגודל שני בתים. בתכנות של 32 סיביות (למשל: Windows 95), משתנה מסוג int הוא בגודל של ארבעה בתים. כדי להיות בטוחים בגודלו של משתנה, ניתן להשתמש באופרטור מיוחד שנקרא sizeof.

האופרטור sizeof יכול לפעול על שני סוגי אופרנדים:

טיפוס של משתנה - למשל: sizeof(float), ואז מתקבל הגודל של משתנה בודד מאותו הטיפוס. ניתן גם לבדוק טיפוסים, שהוגדרו בעזרת typedef (שונה מעט עבור C++).

משתנה שהוגדר בעזרת טיפוס מסוים, יתן את גודל הטיפוס שהגדיר אותו, או מערך יתן את האורך הכולל שתופס המערך בזיכרון, שהוא בעצם גודל הטיפוס של המערך כפול מספר האיברים. למשל: נגדיר ;[3]int arr. הפעלת האופרטור כך - sizeof(arr), תחזיר גודל של int כפול 3 איברים במערך.

קיים סוג מיוחד של מצביע, שמצביע לאיבר שגודלו בית אחד. להבדיל ממצביע ל- char (משתנה בגודל בית אחד), המצביע המיוחד הזה מצביע על תא זיכרון אחד ולא על משתנה מסוג כלשהו. מדובר במצביע ל- void. לא ניתן לבצע פעולות מתמטיות בין מצביעים מסוגים שונים. מצביע ל- void מצביע על כתובת עם טיפוס לא ידוע, ובעצם הוא משתנה מסוג כתובת זיכרון. מכיון שלמצביע ל- void אין טיפוס מוגדר ניתן להוסיף, לחסר, או להציב אותו במצביע מכל סוג שהוא.

המרה - אם אנו רוצים באמת לבצע פעולת מצביעים מוזרה, כמו להשתמש במצביע של int, כדי להצביע על משתנה מסוג long, או אם ברצוננו להציב כתובת ששמורה במצביע מסוג char לתוך מצביע מסוג float, נשתמש בהמרה (casting):

```
long l;
int *iptr;
float *flp;
char *str="hello";

flp=(float*)str;
iptr=(int*)&l;
```

או אפילו כך:

```
l=(long)str;
```

במשתנה str, שמורה הכתובת של המערך "hello". המשתנה l מקבל את הערך ששמור במשתנה str כפי שהוא. אם הכתובת היא 126 אז ערכו של l יהיה 126.

מצביעים לפונקציות

שפת C, כמו גם שפת אסמבלי, מאפשרת להשתמש במצביעים לפונקציות. ניתן לקבל כתובת של פונקציה, ניתן לבצע אליה קריאה, ואפילו ניתן לשמור כתובת כזו במערך או ב- struct.

ההגדרה ()int *func, מצהירה על פונקציה בעלת ערך מוחזר מסוג מצביע ל- int.

ההגדרה ()int (*func), מצהירה על מצביע לפונקציה שמחזירה int.

נראה שימוש במצביע לפונקציה בשני קטעי הקוד הבאים:

הפונקציה ()AskUser מקבלת מספר פעולה, ועל פי המספר, קוראת לפונקציה אחרת, שמבצעת עבודה מול המשתמש.

```
int EnterArrivalTime();
int EnterBreakTime();
int EnterLeaveTime();

int AskUser(int forWhat)
 {
  switch (forWhat)
   {
    case 0: return(EnterArrivalTime());
    case 1: return(EnterBreakTime());
    case 2: return(EnterLeaveTime());
    default: printf("Error. This can't be happening.\n");
   }
  return(0);
 }
```

שימוש בעזרת switch שקול לשימוש במשפטי if מרובים. כל בלוק של case מאלץ השוואה. אם התנאי אינו מתקיים, עוברים לבלוק הבא, עד אשר מוצאים תנאי קיים. אם מדובר במספר רב של בדיקות, הפעולה תארוך זמן רב מאוד.

קטע הקוד הבא משפר את ביצועי התוכנית, אך גוזל מעט יותר זיכרון:

```
int EnterArrivalTime();
int EnterBreakTime();
int EnterLeaveTime();

typedef int (*funcPtr)();

funcPtr funcArray[3]= { EnterArrivalTime, EnterBreakTime,
                        EnterLeaveTime };

int AskUser(int forWhat)
 {
  return(funcArray[forWhat]());
 }
```

מאוד פשוט ויעיל מבחינת זמן הריצה. זכור, לבדוק אם ForWhat הוא בטווח המתאים ואינו גורם לדריסת נתונים, או שפונה לאיבר שאינו במערך.

למי שלא זוכר, המילה השמורה typedef, מגדירה טיפוס חדש (type definition). "הגדר את XXX בעזרת השם YYY", כאשר XXX הוא הצהרה חוקית, ו- YYY הוא השם החדש של ההצהרה. סדר הפרמטרים הפוך מזה של הוראת קדם המעבד #define. בדוגמה לעיל, מופיעים הפרמטרים זה בתוך זה, אך לא רק שההצהרה חוקית אלא היא הדרך להצהיר על מצביע לפונקציה.

ניתן להצהיר על מערך של פונקציות גם באופן הבא:

```
int a();
int b();
int (*funcArray[3])() = {a,b};
```

גודל מצביע

עד כה הסברתי, שמצביע באופן עקרוני הוא בגודל של שני בתים. למעשה, יש אופציה מיוחדת של המהדר שקובעת את מודל הזיכרון. במודל זיכרון Huge למשל, כל המצביעים בגודל של ארבעה בתים. המעבד 8086, פונה אל הזיכרון בשיטת מקטע–היסט (offset–segment). כל אחד מהם בגודל שני בתים. מצביעים בתוך אותו המקטע, זקוקים להיסט בלבד. תוכניות גדולות זקוקות ליותר ממקטע נתונים יחיד, לכן יש צורך בכתובת זיכרון מלאה של מקטע והיסט. הסבר נרחב על מיעון זיכרון, ניתן למצוא בספרי אסמבלי.

לפעמים אנו רוצים למסור כתובת של נתון, או של פונקציה לדרייבר, או אל TSR (למשל, הדרייבר של העכבר, שיכול לקרוא לפונקציה שלנו בכל פעם שנלחץ מקש). כתובת שתימסר לתוכנית אחרת, חייבת להיות כתובת מלאה שכוללת גם את המקטע. כתובת כזו נקראת כתובת רחוקה (far address), ועלינו להגדיר את המצביע ככזה. להלן הגדרות:

```
char far* str="message from above";
char far a;
int far func(FILE far* fptr);
```

המשתנה str הוא מצביע למחרוזת שגודלו 4 בתים ולא 2, לא משנה מה מודל הזיכרון בשימוש, הביטוי &a מחזיר מצביע בגודל של 4 בתים תמיד, והפונקציה func מקבלת מצביע של 4 בתים, והקריאה אליה היא קריאה רחוקה, בכתובת של ארבע בתים, כמו פסיקה. פונקציה כזו יכולה להיות מופעלת על ידי דרייבר (פונקציית callback).

באופן דומה למגדיר far, ניתן להגדיר כתובת כ- near, שפירושה כתובת של שני בתים, ההיסט בלבד. כאשר מודול בשפת C מקושר למודול אחר שנכתב בשפת C או כל שפה אחרת חשוב מאוד לתאם את הצהרת המצביעים בין שני המודולים.

קיימת מילה שמורה נוספת - huge - שפועלת בדיוק כמו far, מלבד הבדל יחיד והוא שמתמטיקה בין שני משתנים כאלה לא תגרום לגלישה, כיון שהם מומרים לכתובת רציפה לפני החישוב, ובחזרה למקטע-היסט אחריו (כתובת של מודל זיכרון flat שבשימוש במצב מוגן 32 סיביות).

קיים סוג נוסף של כתובת בזיכרון, כתובת של מקטע בלבד. ניתן לחבר מקטע וכתובת קרובה (near) כדי לקבל כתובת רחוקה. כדי להגדיר כתובת מסוג מקטע יש להשתמש

במילה השמורה seg_. כתובת מקטע היא בגודל שני בתים. יש לציין כי המילה השמורה seg_ לא בהכרח מוכרת על ידי כל המהדרים.

הרחבה נוספת לשפת C היא cs, _ds, _es, _ss_ בעזרתם ניתן להגדיר שמצביע מסוים מאולץ למקטע מסוים, למשל כך:

```
char _ss *ptr;
```

אך גם בהרחבה זו אין כל התחייבות כי כל מהדר יקבל אותה, ואין לה כל קיום על מחשב שאינו PC (תואם IBM).

תכנות ברמה תחתית (low-level) עד כדי כך דרוש לרוב לתוכניתני אסמבלי, מפתחי דרייברים, ומתכנתים שמקשרים (בשימוש ב- linker) שני קטעי קוד שנכתבו בשפות שונות. למעשה ישנם מספר מגדירים נוספים שקובעים את העברת הפרמטרים ונוהל הקריאה, אך לפני אלו נסקור כיצד משתמש המהדר של שפת C במשאבי המעבד שלרשותו.

ניהול משאבים

הזכרתי מוקדם יותר, כי שפת C פועלת בדומה מאוד למערכת. למעשה פונקציה בשפת C, בנויה על פי אותם הכללים שלפיהם בנויות פסיקות של BIOS, ושל מערכת ההפעלה:

אוגרים כלליים AX, BX, CX, DX הם לשימוש הבלעדי של הפונקציה. אין כל חובה לשמור את ערכם, ולאחר חזרה מקריאה לפונקציה ערכם לא ידוע (שונה עבור פסיקות חומרה, שם ערכם לא משתנה).

אוגרים CS, DS, ES, SS חייבים לחזור בערכם המקורי. אם פונקציה משנה אותם, עליה לשחזר אותם.

משתנים גלובליים נמצאים במקטע DS.

משתנים מקומיים נמצאים במקטע SS, במחסנית. בתחילת טווח-ההכרה שלהם (scope), שלאחר סוגריים מסולסלים, מוקצה זיכרון במחסנית. ביציאה מטווח ההכרה, המשתנה משוחרר מהמחסנית (פקודת אסמבלי pop).

כדי לגשת למשתנים מקומיים, ניתן להשתמש באוגר BP, אך ערכו חייב להיות משוחזר ביציאה מהפונקציה.

פונקציה תקצה משתנים מקומיים, ותשחרר אותם מהמחסנית. פרמטרים מועברים אף הם במחסנית, אבל מכיון ששפת C מכירה בסוג מיוחד של פונקציות, בעלות רשימת פרמטרים משתנה (...), דוגמת ;(...,*printf(char, הרי רק מי שולח את הפרמטרים יודע כמה פרמטרים היו. לכן, פונקציה אינה מסירה מהמחסנית את הפרמטרים שקיבלה, אלא מי שזימן אותה. האוגרים DI ו- SI שמורים אף הם.

ערך מוחזר מועבר באמצעות חצי אוגר AL עבור בית, אוגר AX עבור מילה, והצירוף DX:AX עבור מילה כפולה (ארבעה בתים). שיטה זו תואמת את פסיקות המערכת.

קישור

ניתן לקשר מודול של תוכנית, שנכתב בשפת C, למודול של תוכנית, שנכתב בשפה אחרת. כדי לבצע זאת, בין שאר הדברים, על שתי השפות להסכים על שיטה אחת של העברת פרמטרים וביצוע קריאה לפונקציות. קיימות מספר מילים מיוחדות (שמורות) בשפת C, שמאפשרות להגדיר את שיטת הקריאה לפונקציה:

cdecl, _cdecl, __cdecl : C DECLeration תואם את השיטה של שפת C שהוזכרה לעיל. בנוסף, כל השמות בפונקציה, מקבלים קו תחתי בתחילתם.

pascal : הצהרה תואמת שפת פסקל. הפרמטרים נדחפים לפי סדר הופעתם בקריאה (בשפת C נדחפים מהסוף להתחלה, כך שיישלפו על פי סדר ההופעה), כל השמות באותיות גדולות. הפונקציה אחראית לנקות את הפרמטרים, שהועברו אליה מהמחסנית.

__stdcall : העברת פרמטרים וניקוי מחסנית בדומה ל- cdecl, אבל הקריאה מהירה ממנה. השם של הפונקציה מקבל @_ בתחילתו, ואת מספר הבתים שמועברים בדצימלית בסופו.

למשל, עבור ההצהרה ;int __stdcall func(int b), תתקבל פונקציה בשם @func2_. שיטה זו בשימוש בקריאות מערכת וקריאות DLL של Windows 32 סיביות, או בקישור לשפת Fortran.

__fastcall : העברת פרמטרים, עד כמה שניתן דרך האוגרים. פרמטרים שהועברו דרך המחסנית, מנוקים על ידי הפונקציה עצמה. לפני שם הפונקציה מוסיפים @, אחרי שם הפונקציה מוסיפים @, ואחריו מוסיפים את מספר הבתים שהועברו במחסנית. שימוש במשתנים, שנמצא באוגר, הוא מאוד יעיל בזמן הריצה. אין חובה כי משתנה כזה יועבר רק כפרמטר, והמתכנת יכול להורות למהדר להכניס משתנה מקומי לאוגר.

הגדרות משתנים

register : הוראה למהדר, להכניס את המשתנה לתוך אוגר של המעבד. פעולות עם משתנה זה יהיו מהירות יותר וגם הקוד קצר יותר. באופן כללי, אין טעם להפעיל את ההגדרה על משתנים, כאשר משתמשים במהדר עם אופטימיזציה, כיון שהאופטימייזר מכניס לבד משתנים לאוגרים. כאשר לא משתמשים באופטימייזר, מומלץ להכניס לאוגרים מונים של לולאות for, ובעיקר כאשר עוסקים בעבודה מול זיכרון המסך.

auto : הגדרה כזו מצהירה על משתנה אוטומטי, שמופיע עם כניסה לטווח-הכרה (scope) ונעלם עם יציאה ממנו. זוהי ברירת המחדל ונדיר להשתמש בהגדרה זו.

volatile : ניתן להכניס יציאות I/O למרחב הכתובות. קיימות גם כתובות רבות שהמערכת מטפלת בהם, למשל: כתובת המקש הבא לקריאה מהמקלדת, היא כתובת קבועה שבעזרת מצביע רחוק (far), ניתן לפנות אליה ללא פונקציית מערכת. משתנה שעלול להשתנות על ידי תוכנית אחרת, דרייבר, או על פי קלט מבחוץ, חייב להיות מוגדר כ- volatile. פירוש הדבר הוא, שלא ניתן לשמור את ערכו באוגר זמני, אלא תמיד חייבים לקרוא מתא הזיכרון ולכתוב אל תא הזיכרון.

const ו-**static**, הם שני מגדירים נוספים, הדורשים מעט יותר פירוט.

ניהול התוכנית

שפת תכנות, היא כלי בידיו של המתכנת. על המתכנת להכיר את השפה, את התחביר שלה, את המילים השמורות שבה ואת פונקציות הספריה המובנות (built in), שהיא כוללת. מטרתם של כל ספרי הלימוד, היא ללמד באופן מלא או חלקי, צדדים אלה בשפה. בסעיף זה, אינני מלמד את כלליה של שפת C, אלא מסביר אותם ומראה כיצד על המתכנת להשתמש בכלים אלה, כדי לנצל את היכולות של השפה ולהגיע למימוש המיטבי בה.

הידור

הידור הוא כינוי התהליך, שלוקח קובץ של קוד, שכתב המתכנת (source), והופך אותו לקובץ הרצה (עם סיומת EXE) אותו ניתן להפעיל על המחשב. הידור קובץ source של שפת C מבוצע בשלושה שלבים שונים:

קדם מהדר (Pre-Processor, **קדם-מעבד**), מבצע פעולות הכנה עבור המהדר.

מהדר (Compiler), מתרגם פקודות בשפת C לפקודות מכונה, אבל אינו מחליט על כתובות בזיכרון.

מקשר (Linker), מקבל מספר קבצים לאחר הידור, ומקצה כתובות זיכרון לכל הפונקציות והמשתנים.

קדם המהדר הוא חלק משפת C, ומסתבר שדי נדיר למצוא אותו בחלק משפות אחרות. קדם המהדר מטפל בחזות החיצונית של הקוד, בלי להבין אותו כלל. תפקידיו של קדם המהדר, הם למחוק מהקוד את כל ההערות, כך שהמהדר יקבל קוד מקור נקי, ולטפל בפקודות שמתחילות בסימן # בתחילת שורה. קדם המהדר פועל בלי להבין למה הוא מעתיק, מדביק או מוחק חלקים מהתוכנית.

המהדר הוא המוח שמאחורי שפת C, והוא היחיד בתהליך ההידור שמבין באמת את השפה. קוד המקור (source), שמקבל המהדר, עבר טיפול מוקדם של קדם המהדר שביצע עליו עריכה ראשונית. תפקידו של המהדר הוא להבין את הפקודות ולייצר קוד מכונה, שהמעבד יכול לבצע. המהדר אינו יוצר כתובות, ובמקום זאת מכניס שמות של משתנים לתוך קוד המכונה. הקובץ שיוצר המהדר הוא בעל סיומת OBJ (על שם OBJECT).

המקשר אינו קשור בהכרח לשפת C, תפקידו הוא לקבל מספר קבצי OBJ של שפת תכנות אחת או יותר, ולקשר אותם יחד לתוכנית אחת. תפקידו של המקשר הוא, לייצר כתובות בזיכרון של התוכנית, ולוודא שאין כפילויות בשמות. הקובץ שיוצר המקשר, הוא קובץ הרצה בעל סיומת EXE (executable – בר הרצה).

הצהרות ויישומים

בין שאר תפקידיו הרבים, על המהדר לוודא, שלא השתמשנו במשתנה לא מוגדר. המהדר לא יוכל לבצע על משתנה פעולות מתמטיות אם אינו מוגדר, ולא יוכל לקרוא לפונקציה אם אינו יודע מה הם הפרמטרים שלה. לשם כך, קיימת הצהרת (הגדרת) משתנים, למשל:

```
int a=0;
```

הערה

הצהרה - declaration - הכרזה על משתנה. הגדרה - definition - הצבת ערך ראשוני במשתנה או הצהרה המשולבת בה הצבה.

הצהרה כזו תופיע בקובץ מקור של C, תודיע למהדר על קיומו של המשתנה, על סוג המשתנה, ותודיע לו לבקש מהמקשר לשמור מקום בזיכרון עבורו. המקשר בתורו יראה משתנה חדש בשם a, ויקצה לו כתובת בזיכרון. הבקשה לשמור כתובת חדשה למשתנה a, מופיעה בקובץ ה- Object, שמעביר המהדר למקשר.

אם בשני קבצי Object (מודולים) שונים, קיימת בקשה להגדיר משתנה כזה, המקשר ייצור הודעת שגיאה. כל קובץ מקור (source) בשפת C יוצר קובץ OBJ. אם ברצוננו להשתמש במשתנה יחיד בכל הקבצים, עלינו להודיע למהדר על קיומו ועל הסוג שלו, אך בלי שהמהדר ינסה לשמור לו מקום בזיכרון, אלא במקום זאת יודיע למקשר (Linker), לחפש אותו במודול אחר.

ניקח לדוגמה שני מודולים בשפת C. את המודול האחד כתב חבר שלי לפני חמש שנים. יש בו פונקציות לפתיחת קבצים ואני יודע להשתמש בהן. אם הפונקציה לא מצליחה לפעול, אני מקבל את מספר השגיאה במשתנה גלובלי, שנקרא last_error. במשתנה זה שמור מספר השגיאה האחרונה שהתגלתה. המודול שכולל את הפונקציות, ייראה כך:

```
int last_error=0;
void open_file(char *file_name)
 {
  if (!fopen(file_name,"w+")) last_error=3;
   else last_error=0;
 }
```

המודול שלי, שמשתמש בפונקציות, לא יוכל לכלול את ההגדרה של המשתנה last_error, אבל אם לא נצהיר עליו, המהדר לא יידע איך להשתמש בו כאשר נפנה אליו, ובכלל יחשוב שמדובר בשגיאת כתיב, כיון שהמשתנה לא מוצהר. האמת היא, שקיימת מילה שמורה מיוחדת שאומרת: "ככה מוגדר המשתנה, אבל במודול אחר", והיא extern. לכן, ההצהרה במודול שלי תהיה:

```
extern int last_error;
```

אני לא יכול לאתחל אותו כאן, כיון שלא אני מצהיר עליו. אני רק מודיע למהדר שהוא קיים. המהדר ישתמש במשתנה, וה- Linker (מקשר) יחפש אותו בכל המודולים. אם המשתנה לא נמצא בשום מקום, נקבל הודעה: "Unresolved external in module", שפירושה "משתנה חיצוני לא פתור", או "לא מצאתי את המשתנה שהוגדר בעזרת extern".

כדי לדייק, נבדיל בין **הצהרה** (declaration), לבין **הגדרה** (definition) או **מימוש** (implementation). הצהרה אומרת, שמותר להשתמש באותו השם, וכיצד, ומימוש גם מבקש להקצות לו מקום בזיכרון, וליישם אותו. למעשה, העיקרון מוכר לנו מאוד מפונקציות:

```
void func();
void func()
 {
  ;
 }
```

השורה הראשונה, היא הצהרה בלבד ופירושה הוא "מותר לקרוא לפונקציה הזו ואלה הם הפרמטרים שלה, היא מיושמת במקום אחר". המימוש שמופיע מייד לאחר מכן, מבקש לשמור לפונקציה מקום בזיכרון של הפקודות של התוכנית. אם נצהיר על פונקציה שלא קיימת, ונבצע אליה קריאה, נקבל הודעה דומה לזו שנקבל, אם נשתמש במשתנה שאינו קיים אלא מוצהר בלבד.

קבצי header

קבצי header, הם קבצים שבהם נהוג להכניס הצהרות בלבד. את היישום מכניסים לקובץ C, ונהוג שההבדל היחיד בשם שלהם יהיה הסיומת. למשל לקובץ Fatma.C יהיה קובץ הגדרות Fatma.H. אם קיים קובץ Header (סיומת H), אין צורך להדר שוב את הקובץ עם סיומת C, וניתן להוסיף לפרויקט קובץ OBJ מהודר. למעשה כל פונקציות הספריה של שפת C,שמורות בקבצים כאלה שהמהדר מוצא אותם לבד. הקבצים מכונים קבצי ספריה (Library), והסיומת שלהם היא LIB, אבל עדיין מדובר בסוג של קובץ OBJ. כאשר אנו מכלילים קובץ H של הספריה בעזרת הפקודה include#, המהדר יודע לבד למצוא את קובץ ה-LIB המתאים לו.

אין שום חובה לרשום את ההצהרות בקובץ מיוחד, ומותר לרשום אותן בקובץ C עצמו, שבו הן בשימוש, אך הרבה יותר נוח והרבה יותר מסודר לנהוג כך.

כדי שנוכל באמת לרשום את כל ההצהרות בקובץ יחיד, קיים מנגנון של קדם המהדר והוא ההוראה include#. לאחר ההוראה, יופיע שם קובץ עם נתיב ספריות מלא על הדיסק הקשיח, או חלקי. אם השם מוקף ב- <>, קדם המהדר יחפש את הקובץ בספריה של פונקציות הספריה, ואם השם תחום ב- "", קדם המהדר יחפש בספריה הנוכחית של הפרויקט.

הלכה למעשה, מותר להורות לקדם המהדר לכלול קובץ עם סיומת C, או כל קובץ שהוא בכלל. את קדם המהדר זה לא מעניין בכלל, הוא לוקח את הקובץ החדש, ומעתיק אותו כפי שהוא לתוך הקובץ שמכיל את הפקודה include#, ואז עובר גם עליו.

הוראת קדם מהדר נפוצה נוספת היא define#, אשר מגדירה שם חוקי, כביטוי. למשל:

```
#define EOF -1
```

קדם המהדר יעבור על כל הקבצים (פרט לקבועי מחרוזת ("EOF") ולהערות (/*EOF*/), ובכל מקום שיופיע השם EOF, יוכנס במקומו הערך 1-. כך יישלח הקובץ למהדר, בלי שהוא בכלל יכיר את המילה EOF.

מותר גם להעביר פרמטרים להגדרה כזו:

```
#define add_1(var1) var1++
```

כאן מוגדר, שכאשר מופיע add_1 ובתוך סוגריים משהו, מכניסים במקום כל זה את הביטוי, ואחריו ++. למשל, הפקודה הבאה:

```
add_1("what?");
```

תתורגם ל:

```
"what?"++;
```

חלק מפונקציות הספריה המוכרות ביותר של השפה הם מאקרוס כאלה. להלן דוגמה שימושית יותר מאקרו:

```
#define Print_Fatal(STR) printf("FATAL ERROR: %s\n",STR)
```

עבור:

```
Print_Fatal("out of memory");
```

נקבל:

```
printf("FATAL ERROR: %s\n","out of memory");
```

הבעיה העיקרית עם מאקרוס היא, שקדם המהדר מטפל בהם בלי להבין ממש מה הוא עושה, ואם קיימת שגיאה, היא תהיה ממש לא ברורה. למשל:

```
#define clean(ptr) int a=5; *ptr=0
if (clean(str1)&&clean(str2)) { return(0); }
```

עבור משפט התנאי האחרון, נקבל הודעות שאומרות שהמהדר מצפה לסימן ')' כיון שמופיע לו תו ';' באמצע המשפט, ושלא ניתן להגדיר את a מחדש. וכל זאת על משפט if פשוט. אם היינו משתמשים בפונקציה, המהדר היה יכול לומר לנו במדויק יותר מהי הבעיה.

ניהול קבצי Header

מותר למתכנת להצהיר על פונקציה או משתנה בכל מקום בתוכנית, ומספר רב של פעמים שירצה. המהדר מבין שמדובר באותה ההגדרה על פי השימוש בטיפוס המשתנה, או על פי הפרמטרים של הפונקציה. קדם המהדר אינו בודק פרמטרים, ובעצם מעביר קטע של טקסט שמופיע בתוך סוגריים. כדי למנוע משני מקרוס שונים מלהחזיק באותו השם, קדם המהדר אינו מאפשר להגדיר מאקרו פעמיים, לא משנה מה הם הפרמטרים שלו, ומה הוא היישום שלו. ניתן לבטל הגדרת מאקרו:

```
#undef TRUE
```

undefined, בטל הגדרה. לאחר פקודת קדם מהדר זו, TRUE אינו מוגדר ולא ניתן להשתמש בו. לעומת זאת, מותר להגדיר אותו מחדש לכל ערך שנרצה.

הערה

המאקרו TRUE מוגדר כ- 1 והמאקרו FALSE מוגדר כ- 0. שפת C מגדירה אף את true ואת false. אלה מיושמים כמאקרוס במהדרים הישנים יותר וכחלק מובנה מהשפה בחדשים.

נהוג להכניס את הגדרות כל המאקרוס יחד עם כל ההצהרות לקובץ Header. קובץ כזה יכול להכיל קבצים אחרים נוספים, למשל: הקובץ Dos.h, יכול לכלול בתוכו את StdDef.h (Standard Definition), כדי להשתמש במאקרו NULL. הקובץ IO.h, יכול לכלול אף הוא את הקובץ StdDef.h, ואם התוכנית שלי כוללת את שניהם, אז בעצם הכללתי פעמיים את StdDef.h, והגדרתי פעמיים את NULL, שהוא בעצם דבר אסור.

קיימת לקובץ Header אפשרות לדעת אם הוא כבר נכלל, והמשתנים שלו מוגדרים: הקובץ מגדיר משתנה סתמי ייחודי לו, ובפעם הבאה שיכלילו אותו, הוא יבדוק האם המשתנה מוגדר. אם כן, אז הקובץ כבר הוכלל פעם אחת, ואסור להגדיר את המאקרוס שוב. נבחן את הדוגמה הבאה:

```
#ifndef __STDDEF_H
#define __STDDEF_H
```

אם לא מוגדר __STDDEF_H אז לא היינו בקובץ הזה. לכן, נגדיר את המשתנה, כך שפעם הבאה שנעבור בקובץ, הוא יהיה מוגדר. הגדרה מעשית יותר תיראה כך:

```
#ifndef __STDDEF_H
 #define __STDDEF_H
 #define NULL 0
#endif
```

אם התנאי מתקיים (המאקרו לא מוגדר), אז כל הקטע עד ל- endif# מוכנס לתוכנית. אם התנאי לא מתקיים, פשוט מתעלמים מכל הקטע כאילו שהיה בהערה.

רצוי מאוד להשתמש בנוהל הבא לקבצי Header:

```
#ifndef INCLUDE_MYFILE_H__
 #define INCLUDE_MYFILE_H__

 #include < list of all files >

 #define all defines

 extern char var1;

 void func1();

#endif
```

נדגים עכשיו יישום שימושי מאוד שקיים בשימוש בפיתוח. לשם כך, נשתמש בפונקציה שתקבל מצביע למערך, ותמלא בו את התו '-' עד לסוף המחרוזת (עד תו 0).

```
int Fill_minus(char* str)
 {
  if (0==str) return(0);  // no string this is an error!!!
  while(*str) *str++='-';
  return(1);
 }
```

אפשר להגדיר מאקרו חדש בשם DEBUG, ולהגדיר אותו רק בזמן בדיקות של התוכנית. כאשר אנו משחררים קובץ הרצה לשוק לא נגדיר את המאקרו DEBUG:

```
int Fill_minus(char* str)
 {
#ifdef DEBUG
  if (0==str)
   { puts("Fill_minus:: NULL string!!!"); exit(0); }
#else
  if (0==str) return(0);  // no string this is an error!!!
#endif
  while(*str) *str++='-';
  return(1);
 }
```

אם התוכנית רצה במצב בדיקות, תוצג הודעת שגיאה, והתוכנית תסתיים ("exit()" שבקובץ StdLib.h). השימוש הוא כזה:

```
#define DEBUG         // if we are under debug.
#include "My_Str.h"   // that has this function.

main()
 {
  char* str=(char*)malloc(12);
  Fill_minus(str);
 }
```

אם השתמשנו במערך שהוקצה, בלי לבדוק האם באמת הוקצה זיכרון תחת מצב DEBUG, תוצג הודעה על המסך והתוכנית תסתיים.

נשפר בהרבה את הדיבוג, אם נשתמש במאקרו מובנה. לשם כך, נשתמש בתו '\' בסוף השורה, שמודיע לקדם המהדר שהמאקרו ממשיך גם לשורה הבאה. להלן היישום של המאקרו ASSERT:

```
#define ASSERT(X)     \
 if (0==X) { printf("NULL pointer!!!"); exit(0); }
```

השימוש במאקרו יהיה כך :

```
int Fill_minus(char* str)
 {
  ASSERT(str);
  while(*str) *str++='-';
  return(1);
 }
```

הבעיה עתה היא, שהמאקרו אינו רושם את שם הפונקציה שבה התבצעה השגיאה, לכן אין לנו אפשרות לאתר את מקור ההודעה. במקום להעביר למאקרו את שם הפונקציה שמפעילה אותו, נשתמש במספר מאקרוס שמוגדרים כחלק מהשפה :

__LINE__ - מספר השורה הנוכחית בקובץ (עלול לגרום לבעיות במהדרים מסוימים).

__FILE__ - שם קובץ המקור בשפת C.

__DATE__ - תאריך ההידור.

לכן, הגירסה המלאה של המאקרו תהיה :

```
#ifdef DEBUG
 #define ASSERT(X)                                    \
   if (0==X)                                          \
    {                                                 \
     printf("\nA NULL pointer is used");              \
     printf(" on module %s", __FILE__);               \
     printf(" on line %s.\n", __LINE__);              \
     printf("module version is %s", __DATE__);      \
     exit(0);                                         \
    }
#else
 #define ASSERT(X)
#endif
```

נשתמש בתוכנית הבאה, כדי לבדוק את המאקרו, בהנחה שהמאקרו DEBUG מוגדר בראשית התוכנית :

```
main()
 {
  char *ptr=0;
  ASSERT(ptr);
  return(0);
 }
```

הפלט יהיה :

```
A NULL pointer is used on module ..\PROJECTS\TEST.C on line 8.
module version is May 26 2000
```

מוסכמות

קיימים מספר כללים, שבהם רצוי מאוד להשתמש בתכנות. מומלץ להכיר אותם בכל מקרה, כיון שהספריות עצמן משתמשות בכללים אלו.

כלל ברזל ראשון הוא, שאסור למתכנת לכנות משתנים או פונקציות בשמות שמתחילים בקו תחתי. כל השמות שמתחילים בקו תחתי אחד או שניים שמורים לספריות של השפה. לדוגמה: __STDDEF_H מקובץ StdDef.h, _argv, _argc מקובץ Dos.h, או _fgetc שהיא הפונקציה, שהמאקרו getc קורא לה מקובץ StdIO.h.

אין להשתמש במילים המיוחדות/השמורות הבאות:

```
__asm, __cdecl, __declspec, __export, __far16, __fastcall,
__fortran, __import, __pascal, __rtti, __stdcall, _asm,
_cdecl, __except, _export, _far16, _fastcall, __finally,
_fortran, _import, _pascal, _stdcall, __thread, __try, asm,
auto, bool, break, case, catch, cdecl, char, class const,
const_cast, continue, default, delete, do, double,
dynamic_cast, else, enum, explicit, extern, false, float, for,
friend, goto, if, inline, int, long, mutable, namespace, new,
operator, pascal, private, protected, public, register,
reinterpret_cast, return, short, signed, sizeof, static,
static_cast, struct, switch, template, this, throw, true, try,
typedef, typeid, typename, union, unsigned, using, virtual,
void, volatile, wchar_t, while.
```

חלקן הן מילים שמורות של שפת ++C, או רק של ההרחבות המאוחרות של שפת ++C. בכל מקרה, רצוי מאוד שלא להשתמש בהן, כדי שלא להתרגל לכך, וכדי שהקוד יהיה שמיש בעתיד אם יוחלט להעביר אותו לשפת ++C.

מאקרוס יופיעו תמיד באותיות גדולות.

משתנים ושמות פונקציות, ייכתבו באותיות קטנות. כיום, נהוג להבדיל ביניהם כך שמשתנה מתחיל באות קטנה, ופונקציה מתחילה באות גדולה. נהוג גם לבחור באחת משתי שיטות, לשלב מספר מילים בשמו של המשתנה, האחת על ידי הפרדה עם קו תחתי, והשנייה על ידי שימוש באות גדולה, כך: ()MyFuncIsThis, ()My_func_could_be_this, thisIsMyVar, this_one_too.

נהוג להצהיר על משתנים לפני הצהרה על פונקציות, כיון שלרוב פונקציות משתמשות במשתנים, וזהו הסדר שלהן גם במימוש ולא רק בהצהרה.

אם דרוש שם ייחודי, נהוג להשתמש בשם שמתחיל באות ומסתיים בקו תחתי, לדוגמה, _myVar. באופן מקביל, נהוג שלא להשתמש במשתנים שמסתיימים בקו תחתי באופן שגרתי, כדי שלא להתנגש במשתנים שמתכנתים מנסים "להסתיר".

פונקציות לא יוגדרו כמחזירות void. פונקציה תמיד תחזיר ערך, שמציין האם הפעולה הצליחה או נכשלה. מי שמזמן (זימון=קריאה) את הפונקציה, אינו חייב להשתמש בערך המוחזר, אך פעמים רבות רצוי לבדוק אותו. פונקציות ספריה שונות של עבודה עם קבצים, הדפסה וקריאת נתונים, אינן מבטיחות הצלחה, לכן רצוי מאוד לבדוק את הערך שמוחזר מהן.

שיטות ההחזרה העיקריות:

0 בכישלון, או ערך שונה מ- 0 בהצלחה, לדוגמה: מספר הבתים שנשלחו/התקבלו, או פשוט 1 שמסמל הצלחה.

0 בכישלון, או מצביע לכתובת קיימת בזיכרון בהצלחה, לדוגמה: ()malloc או ()fopen.

0-255 בהצלחה, או int שערכו 1- (65535) בכשלון, מדובר בפונקציות שאמורות לקרוא מקובץ בית שערכו נע בין 0 ל- 255.

כעזר קיימים המאקרוס EOF ו- NULL.

מודולריות

כאשר שני מתכנתים עובדים על תוכנית אחת, או כאשר מתכנת מכין פונקציות לשימוש בעתיד, יש לשמור על מספר כללי מודולריות בסיסיים.

מודול הוא צירוף של קובץ C וקובץ H תואם לו. ברוב המכריע של המקרים, לשניהם אותו שם ורק הסיומות שונות.

מתכנן המודול יצהיר על הפונקציות אותן הוא מייצא (לשימוש מודולים אחרים), והצהרת משתנים מיוצאים, תתבצע בעזרת המילה השמורה extern.

ניתן למסור ערך של משתנה בעזרת פונקציה:

```
char var1_=5;
char var1()
 {
  return(var1_);
 }
```

בקובץ ה- Header, תהיה הצהרה על הפונקציה, אבל לא על המשתנה. באופן זה מודול אחר יכול לקבל את ערך המשתנה, אבל לא לשנות אותו.

למעשה, מודול אחר יכול להצהיר על המשתנה כ- extern ולהשתמש בו. כהגנה קיימת המילה השמורה static. כאשר משתמשים במילה static על משתנה גלובלי, המשתנה מוכר רק במודול בו הוא מוגדר, ולא ניתן לפנות אליו מאף קובץ אחר.

ניתן גם לשמור משתנה גלובלי כך, שיהיה פרטי לפונקציה אחת בלבד:

```
int Counter()
 {
 static int count=0;
 return(count++);
 }
```

המשתנה count הוא גלובלי, לכן הוא מאותחל ל- 0 רק פעם אחת, בתחילה הריצה של התוכנית. הפונקציה היא היחידה שיכולה לפנות אל המשתנה, וכל מה שהיא עושה הוא להחזיר את מספר הפעמים שזימנו אותה פחות 1.

מתכנת יכול גם להגן על עצמו מפני שגיאות תכנות. הסברתי שעדיף לכתוב (5==a) ולא (a==5), למקרה שבו נרשום בטעות סימן שוויון יחיד, אז מתבצעת השמה אל מספר קבוע שהיא שגיאה. כאשר מדובר בשני משתנים, אין ערך קבוע, לכן משתמשים במילה השמורה const.

```
Int print(const char* str)
 {
 static char* lastStr=0;
 if (str==lastStr) return(1);
 lastStr=str;
 return(0);
 }
```

הפונקציה מקבלת מחרוזת ובודקת האם הכתובת שקיבלה היא אותה הכתובת שהתקבלה בקריאה הקודמת. הפרמטר str אינו ניתן לשינוי.

כך ניתן להגדיר מצביע למשתנה קבוע:

```
char const * ptr="constant string";
const char * ptr="constant string";
```

אין הבדל בין שני המשפטים. בשניהם אסור לבצע את ההשמה הבאה: *ptr=0, אבל מותר להציב: ptr=0.

אסור לכתוב שילוב בין שתי הפקודות הללו, כלומר:

```
const char const * ptr="constant string";
```

כיון שזו חזרה על אותה בקשה של const. באופן זה אנו מגינים על הערך ש-ptr מצביע עליו. כדי להגן על ptr עצמו, יש לרשום את המילה השמורה const מייד לפני שם המשתנה, אחרי סימן הכוכבית (*).

כאשר * נכתב לפני const, כך:

```
char * const ptr="constant string";
```

אסור לבצע ptr=0 ומותר לבצע *ptr=0.

כיון שזו בקשה שונה של const, מותר לכתוב:

```
const char * const ptr="constant string";
```

או לחילופין:

```
char const * const ptr="constant string";
```

והמשמעות היא שאסור לבצע ptr=0 וגם אסור לבצע ‎*ptr=0.

כפי שראיתם, ההבדל הוא במיקום const ביחס לכוכבית.

מיתוס

קיים מיתוס, שמתרוצץ בעולם המחשבים כבר מספר שנים. מחברי השפה אף חיברו ספר שמפרט את הכללים שבה. בספר מופיעה הדוגמה הבאה:

```
int main(int argc, char* argv[], char* env[])
 {
  return(0);
 }
```

מתכנתים ואיתם הספרות וקבצי העזרה, אמצו את שמות המשתנים הללו שמשמעותם: מספר ארגומנטים, ערך הארגומנטים וסביבה. למעשה, בקובץ Dos.h, הוכנסו שני משתנים בשמות argv_ ו- argc_, כך שאין צורך להגדיר אותם כפרמטרים של ()main. אולם, כאשר אנו מגדירים אותם כפרמטרים, מותר לכנותם בכל שם שנבחר. הדוגמה הבאה חוקית:

```
int main(int Moshe, char** Itzhak)
 {
  if (!Moshe) puts("no argument count.");
   else printf("first argument value is : %s",
                                      Itzhak[0]);
  return(0);
  }
```

טיפים

מעט טיפים לדרך...

אזכיר כי לא כל מה שנכתב בנספח זה, מומלץ או מותר במסגרת בית הספר.

אני ממליץ מאוד לסכם בשתי מילים וחצי, מה תפקידו של כל מודול בתחילתו. חשוב מאוד לרשום ליד כל פונקציה מה היא מחזירה בהצלחה ומה בכישלון, וערכים חוקיים לפרמטרים אם יש הגבלה כלשהי.

תכנות הוא מקצוע מאוד תובעני, שדורש שעות רבות של תרגול ועבודה. עם זאת הייתי ממליץ מאוד שלא לעבוד תחת עייפות. שגיאה אחת במצב עייפות, תגרור שעות רבות של דיבוג.

קיים איסור על שימוש בתוכנות לא חוקיות, לכן אף אחד מאיתנו לא משתמש בהן. ניתן לתרגל ללא תשלום בבתי הספר הפרטיים לתכנות, כל שיש לעשות הוא, לברר היכן כיתת התרגול, ניתן להיכנס ללא הפרעה, ולקבל גישה למחשבים עם תוכנות פיתוח מתקדמות וחוקיות.

בעולם המחשבים של היום, לא קיים מתכנת אחד שמכיר את כל השפות, את כל מערכות ההפעלה או את כל הפקודות בשפה כלשהי. מומלץ מאוד לרכוש היגיון של מתכנתים, ואז לנסות ולהתמקצע בתחום יחיד. גישה זו תעלה את ערככם בשוק, כאשר תפנו למצוא עבודה בתחום.

תודה מיוחדת לאורן תירוש על הביקורת הבונה, ותרומתו בכתיבת פרק זה.

תודה גם למשה יצחקי ולזיו איילון.

בברכת דרך צלחה

אסף שלי

נספח ב'

טבלאות ASCII, מקשי בקרה ומקשי סריקה

טבלאות ASCII רגיל ומורחב

הערה: שים לב שאת התווים שנמצאים בטבלת **ASCII** המורחבת החל מקוד 127, אפשר להציג על ידי הקלדה של ספרות + Alt. את הספרות יש להקיש במקלדת הנומרית מימין, כאשר ה- Num Lock במצב דלוק.

ערך DEC	ערך HEX	Ctrl	קיצור	משמעות		ASCII
00	00		<NuL>	Null	תו ריק	null
01	01	^A	<SOH>	Start of Header	תחילת ראש התקשורת	☺
02	02	^B	<STX>	Start of Text	תחילת טקסט	●
03	03	^C	<ETX>	End of Text	סיום טקסט	♥
04	04	^D	<EOT>	End OF Trans	סיום תקשורת	♦
05	05	^E	<ENQ>	Enquiry	מה המצב בצד השני של הקו	♣
06	06	^F	<ACK>	Positive Acknowledge	התקבלו נתונים תקינים	♠
07	07	^G	<BEL>	Bell	הפעל פעמון (צפצוף למפעיל)	●
08	08	^H	<BS >	BackSpace (BS)	הזז ראש כותב צעד אחורה	█
09	09	^I	<HT>	Horizontal Tab	הקפץ ראש כותב אופקית קדימה	○
10	A	^J	<NL> <LF>	Line Feed or New Line	התקדם שורה	■
11	B	^K	<VT>	Vertical Tab	קפוץ קדימה מספר שורות	♂
12	C	^L	<FF>	Form Feed	קפוץ דף שלם קדימה	♀
13	D	^M	<CR>	Carriage Return	החזר ראש כותב לתחילת שורה	♪
14	E	^N	<SO>	Shift Out	צא ממצב תקשורת	♫
15	F	^O	<SI>	Shift In	חזור למצב תקשורת	☼
16	10	^P	<DLE>	Data Link Escape	צא מהעברת נתונית בינארית	►
17	11	^Q	<DC1> <Xon>	Device Control 1	המשך בשליחת נתונים	◄
18	12	^R	<DC2>	Device Control 2	שליטה בהתקן מס' 2	↕
19	13	^S	<DC3> <Xoff>	Device Control 3	עצור שליחת נתונים	‼
20	14	^T	<DC4>	Device Control 4	שליטה בהתקן מס' 4	¶
21	15	^U	<NAK>	Negative Acknowledge	התקבלו נתונים שגויים	§
22	16	^V	<SYN>	Syncronous Idle	סינכרון בין צדדים ללא תקשורת	-
23	17	^W	<ETB>	End of Trans Block	סיום העברת בלוק נתונים	↨
24	18	^X	<CAN>	Cancel	בטל הודעה אחרונה	↑
25	19	^Y	<EM>	End of Medium	סיום עבודה בערוץ תקשורת זה	↓
26	1A	^Z	<SUB> <ETF>	Substitute	עבור לערוץ תקשורת חלופי	→
				End of Text File	סיום קובץ טקסט	
27	1B		<ESC>	Escape	חזור לרמה קודמת בסביבת העבודה שלך	←
28	1C		<FS>	File Separator	מפריד בין שני קבצים בתקשורת	FS
29	1D		<GS>	Group Separator	מפריד בין קבוצות של קבצים	GS
30	1E		<RS>	Record Separator	מפריד בין רשומות	RS
31	1F		<US>	Unit Separator	מפריד בין יחידות	US
32	20		<SP>	Space	רווח של תו אחד	SP

Decimal	Hex	Octal	Binary	ASCII
33	21	041	00100001	!
34	22	042	00100010	"
35	23	043	00100011	#
36	24	044	00100100	$
37	25	045	00100101	%
38	26	046	00100110	&
39	27	047	00100111	'
40	28	050	00101000	(
41	29	051	00101001	)
42	2A	052	00101010	*
43	2B	053	00101011	+
44	2C	054	00101100	,
45	2D	055	00101101	-
46	2E	056	00101110	.
47	2F	057	00101111	/
48	30	060	00110000	0
49	31	061	00110001	1
50	32	062	00110010	2
51	33	063	00110011	3
52	34	064	00110100	4
53	35	065	00110101	5
54	36	066	00110110	6
55	37	067	00110111	7
56	38	070	00111000	8
57	39	071	00111001	9
58	3A	072	00111010	:
59	3B	073	00111011	;
60	3C	074	00111100	<
61	3D	075	00111101	=
62	3E	076	00111110	>
63	3F	077	00111111	?
64	40	100	01000000	@
65	41	101	01000001	A
66	42	102	01000010	B
67	43	103	01000011	C
68	44	104	01000100	D
69	45	105	01000101	E
70	46	106	01000110	F
71	47	107	01000111	G
72	48	110	01001000	H
73	49	111	01001001	I
74	4A	112	01001010	J
75	4B	113	01001011	K
76	4C	114	01001100	L

Decimal	Hex	Octal	Binary	ASCII
77	4D	115	01001101	M
78	4E	116	01001110	N
79	4F	117	01001111	O
80	50	120	01010000	P
81	51	121	01010001	Q
82	52	122	01010010	R
83	53	123	01010011	S
84	54	124	01010100	T
85	55	125	01010101	U
86	56	126	01010110	V
87	57	127	01010011	W
88	58	130	01011000	X
89	59	131	01011001	Y
90	5A	132	01011010	Z
91	5B	133	01011011	[
92	5C	134	01011100	\
93	5D	135	01011101	]
94	5E	136	01011110	^
95	5F	137	01011111	_
96	60	140	01100000	`
97	61	141	01100001	a
98	62	142	01100010	b
99	63	143	01100011	c
100	64	144	01100100	d
101	65	145	01100101	e
102	66	146	01100110	f
103	67	147	01100111	g
104	68	150	01101000	h
105	69	151	01101001	i
106	6A	152	01101010	j
107	6B	153	01101011	k
108	6C	154	01101100	l
109	6D	155	01101101	m
110	6E	156	01101110	n
111	6F	157	01101111	o
112	70	160	01110000	p
113	71	161	01110001	q
114	72	162	01110010	r
115	73	163	01110011	s
116	74	164	01110100	t
117	75	165	01110101	u
118	76	166	01110110	v
119	77	167	01110111	w
120	78	170	01111000	x

Decimal	Hex	Octal	Binary	ASCII
121	79	171	01111001	y
122	7A	172	01111010	z
123	7B	173	01111011	{
124	7C	174	01111100	¦
125	7D	175	01111101	}
126	7E	176	01111110	~
127	7F	177	01111111	
128	80	200	10000000	א
129	81	201	10000001	ב
130	82	202	10000010	ג
131	83	203	10000011	ד
132	84	204	10000100	ה
133	85	205	10000101	ו
134	86	206	10000110	ז
135	87	207	10000111	ח
136	88	210	10001000	ט
137	89	211	10001001	י
138	8A	212	10001010	ך
139	8B	213	10001011	כ
140	8C	214	10001100	ל
141	8D	215	10001101	ם
142	8E	216	10001110	מ
143	8F	217	10001111	ן
144	90	220	10010000	נ
145	91	221	10010001	ס
146	92	222	10010010	ע
147	93	223	10010011	ף
148	94	224	10010100	פ
149	95	225	10010101	ץ
150	96	226	10010110	צ
151	97	227	10010111	ק
152	98	230	10011000	ר
153	99	231	10011001	ש
154	9A	232	10011010	ת
155	9B	233	10011011	ø
156	9C	234	10011100	£
157	9D	235	10011101	Ø
158	9E	236	10011110	L
159	9F	237	10011111	Ł
160	A0	240	10100000	á
161	A1	241	10100001	í
162	A2	242	10100010	ó
163	A3	243	10100011	ú
164	A4	244	10100100	ñ
165	A5	245	10100101	Ñ

Decimal	Hex	Octal	Binary	ASCII
166	A6	246	10100110	ð
167	A7	247	10100111	Ð
168	A8	250	10101000	¿
169	A9	251	10101001	ã
170	AA	252	10101010	Ã
171	AB	253	10101011	ℓ
172	AC	254	10101100	ħ
173	AD	255	10101101	¡
174	AE	256	10101110	³
175	AF	257	10101111	¤
176	B0	260	10110000	░
177	B1	261	10110001	▒
178	B2	262	10110010	▓
179	B3	263	10110011	│
180	B4	264	10110100	┤
181	B5	265	10110101	╡
182	B6	266	10110110	╢
183	B7	267	10110111	╖
184	B8	270	10111000	╕
185	B9	271	10111001	╣
186	BA	272	10111010	║
187	BB	273	10111011	╗
188	BC	274	10111100	╝
189	BD	275	10111101	╜
190	BE	276	10111110	╛
191	BF	277	10111111	┐
192	C0	300	11100000	└
193	C1	301	11000001	┴
194	C2	302	11000010	┬
195	C3	303	11000011	├
196	C4	304	11000100	─
197	C5	305	11000101	┼
198	C6	306	11000110	╞
199	C7	307	11000111	╟
200	C8	310	11001000	╚
201	C9	311	11001001	╔
202	CA	312	11001010	╩
203	CB	313	11001011	╦
204	CC	314	11001100	╠
205	CD	315	11001101	═
206	CE	316	11001110	╬
207	CF	317	11001111	╧
208	D0	320	11010000	╨
209	D1	321	11010001	╤

Decimal	Hex	Octal	Binary	ASCII
210	D2	322	11010010	╥
211	D3	323	11010011	╙
212	D4	324	11010100	╘
213	D5	325	11010101	╒
214	D6	326	11010110	╓
215	D7	327	11010111	╫
216	D8	330	11011000	╪
217	D9	331	11011001	┘
218	DA	332	11011010	┌
219	DB	333	11011011	█
220	DC	334	11011100	▄
221	DD	335	11011101	▌
222	DE	336	11011110	▐
223	DF	337	11011111	▀
224	E0	340	11100000	α
225	E1	341	11100001	β
226	E2	342	11100010	Γ
227	E3	343	11100011	π
228	E4	344	11100100	Σ
229	E5	345	11100101	σ
230	E6	346	11100110	µ
231	E7	347	11100111	τ
232	E8	350	11101000	Φ
233	E9	351	11101001	θ
234	EA	352	11101010	Ω
235	EB	353	11101011	δ
236	EC	354	11101100	∞
237	ED	355	11101101	φ
238	EE	356	11101110	∈
239	EF	357	11101111	∩
240	F0	360	11110000	≡
241	F1	361	11110001	±
242	F2	362	11110010	≥
243	F3	363	11110011	≤
244	F4	364	11110100	⌠
245	F5	365	11110101	⌡
246	F6	366	11110110	÷
247	F7	367	11110111	≈
248	F8	370	11111000	°
249	F9	371	11111001	•
250	FA	372	11111010	·
251	FB	373	11111011	√
252	FC	374	11111100	ⁿ
253	FD	375	11111101	²
254	FE	376	11111110	■
255	FF	377	11111111	

טבלת מקשי בקרה (Control Key Code)

עשרוני	הקסדצימלי	אוקטלי	בינארי	מקש
72	48	110	01001000	Up
80	50	120	01010000	Down
75	4B	113	01001011	Left
77	4D	115	01001101	Right
82	52	122	01010010	Insert
83	53	123	01010011	Delete
71	47	107	01000111	Home
79	4F	117	01001111	End
73	49	111	01001001	Page Up
81	51	121	01010001	Page Down
119	77	167	01110111	Ctrl + Home
117	75	165	01110101	Ctrl + End
132	84	204	10000100	Ctrl + Page Up
118	76	166	01110110	Ctrl + Page Down
59	3B	73	00111011	F1
60	3C	74	00111100	F2
61	3D	75	00111101	F3
62	3E	76	00111110	F4
63	3F	77	00111111	F5
64	40	80	01000000	F6
65	41	81	01000001	F7
66	42	82	01000010	F8
67	43	83	01000011	F9
68	44	84	01000100	F10
84	54	124	01010100	Shift + F1
85	55	125	01010101	Shift + F2

עשרוני	הקסדצימלי	אוקטלי	בינארי	מקש
86	56	126	01010110	Shift + F3
87	57	127	01010111	Shift + F4
88	58	130	01011000	Shift + F5
89	59	131	01011001	Shift + F6
90	5A	132	01011010	Shift + F7
91	5B	133	01011011	Shift + F8
92	5C	134	01011100	Shift + F9
93	5D	135	01011101	Shift + F10
94	5E	136	01011110	Ctrl + F1
95	5F	137	01011111	Ctrl + F2
96	60	140	01100000	Ctrl + F3
97	61	141	01100001	Ctrl + F4
98	62	142	01100010	Ctrl + F5
99	63	143	01100011	Ctrl + F6
100	64	144	01100100	Ctrl + F7
101	65	145	01100101	Ctrl + F8
102	66	146	01100110	Ctrl + F9
103	67	147	01100111	Ctrl + F10
104	68	150	01101000	Alt + F1
105	69	151	01101001	Alt + F2
106	6A	152	01101010	Alt + F3
107	6B	153	01101011	Alt + F4
108	6C	154	01101100	Alt + F5
109	6D	155	01101101	Alt + F6
110	6E	156	01101110	Alt + F7
111	6F	157	01101111	Alt + F8
112	70	160	01110000	Alt + F9

עשרוני	הקסדצימלי	אוקטלי	בינארי	מקש
113	71	161	01110001	Alt + F10
120	78	170	01111000	Alt + 1
121	79	171	01111001	Alt + 2
122	7A	172	01111010	Alt + 3
123	7B	173	01111011	Alt + 4
124	7C	174	01111100	Alt + 5
125	7D	175	01111101	Alt + 6
126	7E	176	01111110	Alt + 7
127	7F	177	01111111	Alt + 8
128	80	200	10000000	Alt + 9
129	81	201	10000001	Alt + 0
130	82	202	10000010	Alt + minus
131	83	203	10000011	Alt + equal

תרשים מקשי סריקה רגילים ומורחבים

* שייך ל-External Key Scan Codes (לפניו נשלח הקוד 224)

נספח ג׳

פסיקות עיקריות בחומרה ובתוכנה

מספר פסיקה	תיאור הפסיקה והשימושים (מספר/קוד הפסיקה מסומן בהקסה-דצימלי)
00	חלוקה ב-0. פסיקה זו מתבצעת כאשר מחלקים ב-0.
01	צעד אחר צעד - Single Step. משמשת לתוכניות מסוג Debugger.
02	פסיקה שלא ניתנת למיסוך (NMI). משמשת במחשב לבקרת שגיאות Parity של הזיכרון הראשי במחשב.
03	נקודת שבירה - Break Point, משמשת תוכניות מסוג Debugger.
04	גלישה. מתבצעת כאשר בפעולה חשבונית יש גלישה.
05	הדפסת מסך - Print Screen. תוכן המסך מועבר למדפסת.
06	שמור לצרכים עתידיים.
07	שמור לצרכים עתידיים.
08	פסיקת חומרה של השעון Timer 0.
09	פסיקת חומרה של המקלדת - Keyboard.
0A	פסיקת חומרה ב-AT, שרשור לבקר פסיקות שני. ב-XT פנוי.
0B	פסיקת חומרה לכרטיס תקשורת טורי שני COM2.
0C	פסיקת חומרה לכרטיס תקשורת טורי ראשון COM1.
0D	AT : פסיקת חומרה לכרטיס מקבילי שני LPT2 ; XT : דיסק קשיח.
0E	פסיקת חומרה לכונן דיסקטים.
0F	פסיקת חומרה לכרטיס מקבילי ראשון LPT1.
10	Video BIOS, שירותי מסך.
11	BIOS, שירות לקבלת פירוט מרכיבי החומרה במחשב.
12	BIOS, שירות לקבלת פירוט של גודל זיכרון.

מספר פסיקה	**תיאור הפסיקה והשימושים** (מספר/קוד הפסיקה מסומן בהקסה-דצימלי)
13	BIOS, שירותי גישה לכונני דיסקטים/דיסקים.
14	BIOS, שירותי תקשורת טורית ל-COM2, COM1.
15	BIOS, שירותי כונן קלטות (טייפ), זיכרון Extended.
16	BIOS, שירותי מקלדת.
17	BIOS, שירותי תקשורת מקבילית LPT2, LPT1.
18	BIOS, כניסה ל-Basic הצרוב ב-ROM על הלוח הראשי.
19	BIOS, שירות אתחול למחשב Bootstrep Loader.
1A	BIOS, שירותי זמן : קביעת שעון מערכת, שעון זמן אמת ואזעקה.
1B	BIOS, פסיקה הנגרמת מלחיצה על צירוף המקשים Ctrl+Break.
1C	BIOS, פסיקת שעון המתבצעת 18.2 פעם בשנייה.
1D	BIOS, אתחול מערכת הוידאו לאופני הפעולה שונים.
1E	BIOS, פרמטרים של מערכת כונני הדיסקטים.
1F	BIOS, אתחול טבלת התווים הגרפיים להצגה גרפית.
20	DOS, סיום תוכנית.
21	DOS, שירותים כלליים של מערכת ההפעלה DOS.
22	DOS, כתובת סיום תוכנית. משמשת לחזרה למערכת ההפעלה.
23	DOS, כתובת יציאה לתוכנית בלחיצה על Ctrl+Break.
24	DOS, תוכנית לטיפול בשגיאות קריטיות.
25	DOS, שירות לקריאה ישירה מהדיסק/דיסקט.
26	DOS, שירות לכתיבה ישירה לדיסק/דיסקט.
27	DOS, שירות לסיום תוכנית והשארתה בזיכרון כ-TSR.
28	DOS, תוכנית שירות המציינת ש-DOS במצב מנוחה ומצפה לקלט.
29	DOS, כתיבה מהירה (FAST PUTCHAR).
2A	DOS, ממשק לרשת (Microsoft Network Interface).
2B-2D	DOS, שמור.
2E	DOS, טעינת תוכניות לזיכרון תחת DOS.

מספר פסיקה	תיאור הפסיקה והשימושים (מספר/קוד הפסיקה מסומן בהקסה-דצימלי)
2F	DOS, שירותים מורכבים : שרת הדפסות PRINT, ופרמטרים שונים של תוכניות כמו ASSIGN, SHARE ועוד.
30-32	שמור.
33	פסיקת שירות לטיפול בעכבר.
34-3F	שמור.
40	BIOS, במחשבי AT משמש כהפניה לגישה לכונן דיסק/דיסקט.
41-46	פסיקות BIOS. פרמטרים לגישה לדיסק קשיח במחשבי AT ומעלה.
60-67	שמור לפסיקות משתמש.
50-57	פסיקות חומרה של בקר פסיקות שני במחשבי AT ומעלה.
5C	ממשק ל-BIOS של כרטיס תקשורת (Network BIOS).
67	זיכרון הרחבה Expanded. שירותי Device Driver של הזיכרון.
70-77	פסיקות חומרה של בקר פסיקות שני במחשבי AT ומעלה.
80-85	שמור לשימוש BASIC.
86-EF	שמור לשימוש המפרש - BASIC Interpreter.
F0-FF	פנוי לתוכניות משתמש.

נספח ד׳

כתובות מוחלטות ב-RAM לשימוש BIOS

הכתובת (hex)	שימושים
00000-003FF	מרחב פסיקות. כל פסיקה 4 בתים, סה״כ 256.
00400-00407	כתובות של עד 4 מתאמי תקשורת טורית (COM).
00408-0040F	כתובות של עד 4 מתאמי תקשורת מקבילית (LPT).
00410-00411	פירוט של כל מרכיבי החומרה העיקריים במחשב.
00412	דגל אתחול.
00413-00414	גודל הזיכרון בפועל בכפולות של 1K.
00417-00418	דגלי מצב של המקשים המיוחדים במקלדת.
0041A-0041B	מצביע לראש התור של המקלדת.
0041C-0041D	מצביע לסוף התור של המקלדת.
0041E-0043D	תור למקשים שנלחצו במקלדת וטרם נקראו.
0043E-00448	פרמטרים לחיוויים מכונני הדיסקטים.
00449	אופן פעולה נוכחי של מתאם המסך.
0044A-0044B	מספר העמודות להצגה במסך.
0044C-0044D	כמות הבתים הנחוצה להצגת מסך אקטיבי (Page).
0044E-0044F	כתובת התחלתית במתאם המסך של הדף האקטיבי.
00450-00451	מיקום הסמן בדף 0 במסך.
00452-0045F	מיקום הסמן עבור דפים 1 עד 7.
00460-00461	אופן ההצגה של הסמן על גבי המסך.
00462	כמות דפי הטקסט שניתן להציג על גבי המסך.
00463-00464	כתובת התחלתית במרחב קלט/פלט של כרטיס מסך.
00466	ערך של טבלת הצבעים (Color Palette).
0046C-0046F	4 בתים שמחזיקים את ערך השעון של המערכת.
00470	דגל של שעון מערכת המציין שחלפה שעת חצות.

נספח ה'

אוגרים

קיימים במעבד **14 אוגרים** (registers), אשר כל אחד מהם הינו בגודל של מילה אחת (16 סיביות). כמו בזיכרון, כך גם האוגר מכיל נתון, אשר אפשר לקרוא אותו, או להחליפו באחר. כיבוי המחשב גורם לאובדן הנתונים שהיו באוגר.

האוגרים משמשים לשמירת נתונים, לביצוע פעולות חשבוניות ופעולות לוגיות, להעברת נתונים אל הזיכרון וממנו, ועוד.

לכל אוגר יש תפקיד, אולם בכמה מהאוגרים משתמשים למטרות דומות. ניתן לחלק את האוגרים ל-4 קבוצות, על פי התפקיד שלהם.

אוגרים כלליים (General Purpose registers)

	High	Low	
Accumulator	AH	AL	AX
Base	BH	BL	BX
count	CH	CL	CX
Data	DH	DL	DX

לרשותנו 4 אוגרים כלליים. נבחין שכל אחד מהאוגרים ניתן לחלוקה לשני חצאי אוגר. לכל מחצית אוגר יש סימון של אות זיהוי של האוגר ולידה האות H או L. החלק השמאלי מסומן באות H, כדי לציין High (החלק "הגבוה" של המספר) והחלק הימני מסומן באות L כדי לציין Low (הערך "הנמוך").

1. **אוגר AX** - אוגר זה מכונה גם צובר או אקומולטור (משתמשים בו לסיכום מספרים). החלק השמאלי נקרא AH (H לציון High) והחלק הימני נקרא AL (L עבור Low).
2. **אוגר BX - אוגר הבסיס.** החלק השמאלי נקרא BH והימני BL.
3. **אוגר CX - אוגר המונה**. החלק השמאלי נקרא CH והימני CL.
4. **אוגר DX - אוגר הנתונים**. החלק השמאלי נקרא DH והימני DL.

למעשה, על ידי חלוקה זו קיבלנו 12 אוגרים שונים: ניתן לפנות למשל לאוגר AX, השלם, או לכל אחד משני חצאיו: אוגר AL, או אוגר AH, וכן הלאה.

אוגרי המקטע (Segment registers)

CS	Code Segment
DS	Data Segment
ES	Extra Segment
SS	Stack Segment

5. **אוגר CS** - **אוגר מקטע הקוד**. מכיל את כתובת ההתחלה של התוכנית.
6. **אוגר DS** - **אוגר מקטע הנתונים**. מכיל את כתובת ההתחלה של מקטע הנתונים.
7. **אוגר ES** - **אוגר הנתונים נוספים**. מכיל את כתובת תחילת הנתונים הנוספים (אם ישנם).
8. **אוגר SS** - **אוגר המחסנית**. מכיל את כתובת תחילת המחסנית (יוסבר בהמשך).

אוגרים מצביעים (Pointer registers)

SI	Source Index
DI	Destination Index

9. **אוגר SI** - **אוגר מצביע מקור**. מצביע על כתובת תא זיכרון מבוקש.
10. **אוגר DI** - **אוגר מצביע יעד**. מצביע על כתובת תא זיכרון מבוקש.

BP	Base Pointer
SP	Stack Pointer

11. **אוגר BP** - **מצביע הבסיס**. משמש כמצביע על כתובת תאי הזיכרון במחסנית.
12. **אוגר SP** - **מצביע המחסנית**. מצביע על כתובת תא הזיכרון בקצה המחסנית.

IP	Instruction Pointer

13. **אוגר IP** - **מצביע פקודה**. זוהי הכתובת של ההוראה **הבאה** לביצוע.

אוגר הדגלים (Flags register)

F	Flags Register

14. **אוגר F** - **אוגר הדגלים**. מכיל 9 דגלים (סיביות) הנמצאים במצב אחד או אפס לוגי. הם מתארים מאפיינים מסוימים בפעולה האחרונה שבוצעה על ידי היחידה אריתמטית-לוגית (ALU) של המעבד.

נספח ו'

הרצת תוכניות הדוגמה במהדר Turbo C/C++ 3.x

הערה

מרבית התוכניות בספר זה עוסקות בפסיקות.

דבר זה אפשרי רק בעזרת מהדר התומך בכך (מהדר המותאם לסביבת DOS), **כגון** Borland Turbo C/C++ 3.0 **או** Borland Turbo C/C++ 3.1.

גרסת Lite של Borland (למרות שהיא מיועדת ל-DOS) אינה מתאימה, בין השאר מכיון שהיא אינה תומכת בהקצאות גדולות של זיכרון והידור במודל LARGE הנחוצים להידור הפרויקטים בנושאי המולטימדיה.

הגדרת מאפייני המהדר

תוכניות הדוגמה המצורפות בתקליטור הן ברובן פרויקטים הדורשים משאבי זיכרון נרחבים יותר מברירת המחדל של המהדר. מסיבה זו, עלינו לבצע מספר שינויים בהגדרות המהדר כגון הגדלת מודל הזיכרון מ-Tiny ל-Large והגדלת ה-heap Size ל-640K.

לשם כך, עקבו אחר הוראות הבאות ושנו את ההגדרות כך שחלונות ההגדרות ייראו בדיוק כפי שהן נראות בתמונות שבנספח זה.

אין צורך לשנות הגדרות אלו בכל פעם מחדש! עליכם לשנות את ההגדרות רק בפעם הראשונה עם תחילת קריאת הספר, והמהדר יזכור את השינויים שביצעתם.

1. הגדלת מודל הזיכרון

בחרו בתפריט Code generation > Compiler > Options (תרשים 1).

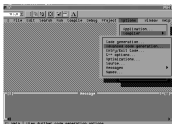
תרשים 1

ושנו את המאפיינים כך שייראו כפי שהם נראים בתרשים 2.

תרשים 2

2. הגדלת ה-Far Data Threshold ל-64000

בחרו בתפריט Options > Compiler > Advanced code generation (תרשים 3).



תרשים 3

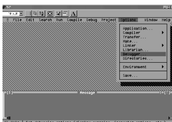
ושנו את המאפיינים כך שייראו כפי שהם נראים בתרשים 4.



תרשים 4

3. הגדלת ה-Heap Size ל-640

בחרו בתפריט Debugger < Options (תרשים 5).



תרשים 5

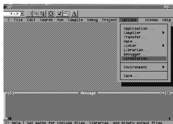
ושנו את המאפיינים כך שייראו כפי שהם נראים בתרשים 6.



תרשים 6

4. עדכון מיקום הספריות

כדי שהמהדר יוכל למצוא את קבצי יחידות הספריה, יש לוודא שהוא יידע מהו מיקומם המדויק. לשם כך, בחרו בתפריט Options < Directories (תרשים 7).



תרשים 7

ודאו ששני השדות הראשונים מכילות את המיקום הנכון. אם, לדוגמה, המהדר שלכם מותקן במיקום "C:\TurboC" השדות צריכים להיראות כפי שהם נראים בתרשים 8.



תרשים 8

פתיחת תוכניות הדוגמה

לפני שנוכל להדר את תוכניות הדוגמה, עלינו להעתיק את תוכן הספריה Books\59281 מהתקליטור המצורף לספר זה, אל הדיסק הקשיח.

היכן נמצאים הקבצים הקשורים לספר זה?

התיקיה הרלוונטית לספר זה: Books\59281

תחת תיקיה **X:\Books\59281** (החלף את האות X באות הכונן המתאימה) תמצא תיקיות משנה: תיקיה עבור כל פרק: תיקיה 01 עבור פרק 1, תיקיה 02 עבור פרק 2 וכך הלאה.

תוכניות הדוגמה של פרקים: 1, 2, 3, 5, 6, 7 ו-17 הן תוכניות רגילות ולכן נפתח אותן כתוכניות רגילות בשפת C.

תוכניות הדוגמה של פרקים: 4, 8, 9, 10, 11, 12, 13, 14, 15 ו-16 הם **פרויקטים** המשלבים יחידות (Units) וקובץ ראשי המאחד בין היחידות (אשר תוכנו הודפס בספר). לכן, עלינו לפתוח אותם כפרויקטים.

העתקת קבצי המקור לדיסק

הקבצים נמצאים בתיקיה **X:\books\59281** מסודרים לפי תיקיות - תיקיה עבור כל פרק. כדי להעתיק את הקבצים לדיסק:

1. לחצו על לחצן **התחל**, **תוכניות**, **סייר Windows**.
2. הציגו את תוכן התיקיה Books אשר בתקליטור.
3. סמנו את התיקיה **59281** וגררו אותה לתיקיה כלשהי בדיסק C (למשל BookC).

מכיון שמקור הקבצים הוא התקליטור, הם מסומנים לקריאה בלבד. רצוי לשנות מאפיין זה בדרך זו:

1. דרך הסייר היכנסו לתיקיה בדיסק, שבה נמצאים הקבצים שהעתקתם.
2. סמנו קובץ מסוים או את כולם על-ידי Ctrl+A.
3. הציבו את סמן העכבר מעל האזור המסומן ולחצו לחיצה ימנית בעכבר.
4. מתפריט הקיצור בחר **מאפיינים**.
5. בטלו את הסימון בתיבה **קריאה בלבד** (דאגו שתיבה זו תהיה ריקה).
6. לחצו על **החל**, לחצו על **אישור**.

אפשרות אחרת:

1. לחצו על לחצן **התחל**, **תוכניות**, **הפנייה ל-MS-DOS**.
2. בחלון DOS רשמו: attrib -r +a c:*directory_name**.* /s.

שימו לב, כל הפעולות שהוזכרו עד כה הן פעולות שעלינו לבצע באופן חד פעמי כדי להתאים את המהדר לתוכניות הדוגמה הדורשות זיכרון נרחב, וכדי לאפשר למהדר להדר את התוכניות בכונן המאפשר כתיבה (ולא קריאה בלבד כמו כונן התקליטורים).

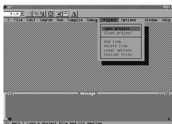
בהמשך נלמד מהן הפעולות שעלינו לבצע בכל פעם שנרצה לפתוח תוכנית לדוגמה.

כדי להריץ אחת מהתוכניות הללו עליכם לבצע את הפעולות הבאות:

1. לפתוח פרויקט חדש בתוך ספריית הפרק המתאים

שימו לב! חשוב שכל הקבצים של הפרויקט יימצאו באותה ספריה בה אנו פותחים את הפרויקט ולכן הקפידו לציין את מיקום ספריית הפרק המתאים.

נעשה זאת בעזרת תפריט Project < Open Project (תרשים 9).



תרשים 9

אחר כך נקליד את מיקום ושם הפרויקט. לדוגמה אם העתקנו את תוכן הספריה Books\59281 מהתקליטור למיקום "C:\BookC" ואנו רוצים לפתוח את תוכנית הדוגמה של פרק 10, נקליד: C:\bookC\10\proj.prj (תרשים 10).

תרשים 10

2. לצרף לפרויקט שפתחנו את הקבצים הנחוצים

נבחר בתפריט Project > Add Item (תרשים 11).

תרשים 11

בתיבה שתיפתח נבחר בכל קבצי הקוד הנחוצים (על פי הרשימה שבסוף כל פרק תחת הכותרת "הקבצים שיש לצרף לפרויקט כדי להריץ את התוכנית לדוגמה"), ונלחץ OK.

נספח ז'

התקליטור המצורף

קרא את קובץ ONCD.DOC בתקליטור כדי לקבל מידע מעודכן על תכולת התקליטור המצורף.
קרא בנספח ו' כיצד להעתיק את קבצי המקור הרלוונטים לספר זה.

- **קטלוג HTML** - קטלוג ספרי המחשבים האינטראקטיבי של **הוצאת הוד-עמי**. לשם קריאת הפרקים לדוגמה יש להתקין את תוכנת Adobe Acrobat Reader אשר מצורפת בתקליטור. הוראות התקנה בהמשך.
  הקטלוג מומלץ לצפייה באמצעות Internet Explorer גירסה 5, המצורפת בתקליטור. הוראות התקנה בהמשך.
  התקנת שתי התוכנות קלה וניתנת לביצוע באמצעות קישור ישירות מהקטלוג.
- מספר תוכנות עזר שימושיות (תחת התיקיה SoftWare).
- קטעי התוכניות שבספר.



הערה!

אם מנהל התקן כונן התקליטורים המותקן הוא 16 סיביות - ייתכן שתראה רק 8 תווים ראשונים של שם קובץ (במידה שהמקור ארוך יותר).
הסיבה: כונני תקליטורים במהירות x4 עובדים עם מנהל התקן שעבד בסביבת DOS ו-Windows 3.11 ויכול לעבוד גם עם Windows 95, למעט היכולת לזהות קבצים עם שמות ארוכים.
הפתרון: להתקין מנהל התקן 32 סיביות (אם קיים), או לקנות כונן תקליטורים חדש ולוודא שמצורף אליו מנהל התקן 32 סיביות.

התיקיה הרלוונטית לספר זה

שים לב: בתיקיה **Books** יש מספר תיקיות המכילות קבצי תרגול ל**ספרים בהוצאת הוד-עמי**. שם התיקיה בנוי ממספר הדאנאקוד הנמצא בעטיפה האחורית של הספר (לדוגמה: דאנאקוד 259-10169 יופיע בתיקיה כ- 59169).

התיקיה הרלוונטית לפרק זה:

Books\59281

ראה הוראות בנספח ו' כיצד להעתיק את הקבצים לדיסק הקשיח וכיצד להריץ את התוכניות.

Acrobat Reader - התקנה

יש להתקין תוכנה זו כדי לקרוא ולהדפיס את הפרקים לדוגמה, אליהם ניתן לגשת באמצעות **קטלוג HTML** (שהתקנתו תוסבר בהמשך). התוכנה גם מאפשרת חיפוש בעברית ובאנגלית במסמך המוצג. בנוסף, בעזרת תוכנה זו תוכל לקרוא את המסמכים שהוצאה מפרסמת באתר האינטרנט. התוכנה פועלת במערכות הפעלה **Windows 95 ומעלה!**

1. לחץ על לחצן **התחל** ובחר באפשרות **הפעלה**.
2. בתיבת הטקסט הקלד את הפקודה **X:\Software\Adobe Acrobat\Arme4ENU.exe** (החלף את האות X באות המייצגת את כונן התקליטורים שלך) ולחץ על **אישור**.
3. אשף ההתקנה מתקין את הרכיבים הנדרשים. עליך ללחוץ על **Next**, **Accept** ו-**Next** פעם נוספת כדי לסיים את ההתקנה.
4. בסיום ההתקנה עשויה להופיע על המסך תיבת דו-שיח **התנגשות בין גירסאות** ומייד אחר כך להיעלם. במקומה תופיע על המסך תיבת הודעה של תוכנית ההתקנה. לחץ על **אישור** ובתיבת הדו-שיח **התנגשות בין גירסאות** ששבה להופיע לחץ על **כן**, כדי לשמור את גרסת הקובץ שלך.

קטלוג HTML

הוצאת הוד-עמי גאה לבשר על **קטלוג HTML** העושה שימוש בטכנולוגיות אינטרנט מתקדמות כדי להביא לך את המידע על ספרי המחשבים המקצועיים שלנו בלחיצת עכבר.

מומלץ לצפייה בעזרת Microsoft Internet Explorer מגירסה 5 ומעלה.

בעזרת **קטלוג HTML** תוכל:

- לעיין במידע על ספרי ההוצאה מתי שתרצה (לחיצה כפולה.... וזהו!).
- לעבור במהירות ובקלות בין הקטלוג והיישום בו אתה עובד.
- לעיין במידע על כל ספר וספר.
- לצפות ואף להדפיס פרק לדוגמה.
- לגעת במהירות, בגישה אינטואיטיבית, תוך התמקדות מהירה בספר המבוקש.
- לעיין בקטלוג בקצב אישי שלך.
- לנווט את דרכך בקטלוג ולחזור ולהתרענן בכל נושא בכל רגע.



קטלוג ספרי מחשבים בהוצאת הוד-עמי

הקטלוג מומלץ לצפייה בעזרת Internet Explorer מגירסה 5 ומעלה.

1. הכנס את התקליטור לכונן.
2. לחץ **התחל** ובחר **הפעלה**.
3. בעזרת לחצן עיון סמן את הקובץ **Setup.exe** אשר בתיקיה הראשית של התקליטור המצורף.
4. לחץ **פתח**.
5. לחץ **אישור**.

המחירון המעודכן של ספרי ההוצאה נמצא באתר האינטרנט www.hod-ami.co.il

1. ודא שתקליטור הוד-עמי נמצא בכונן התקליטורים.
2. הפעל את הסמל עם הכיתוב **קטלוג ספרי מחשבים בהוצאת הוד-עמי** שעל שולחן העבודה.

מה עוד בתקליטור?

הוצאת **הוד-עמי** מפיצה תוכנות אלו כבונוס ללקוחות ההוצאה, ואינה מתיימרת לגבות תשלום עבור התוכניות המצורפות ו/או לתמוך בהם.



אזהרה:

השימוש בתקליטור זה הוא על אחריותו הבלעדית של המשתמש. המוצרים המותקנים בתקליטור זה מסופקים באחריות החברות המייצרות אותם. הוצאת **הוד-עמי** אינה אחראית, בכל צורה שהיא, לאופן ולטיב התוכנות המותקנות.

בכל שאלה לגבי תוכנה הנמצאת בתקליטור, יש לפנות למפתחי התוכנה (כל תוכנה בנפרד) כפי שמצוין בקבצי העזרה של התוכנה המדוברת.

הקבצים הם גרסאות **שיתופיות** (ShareWare) **וחופשיות** (FreeWare).

גרסת ShareWare מאפשרת לך, המשתמש, לבדוק את יעילות התוכנה ואת תאימותה לעבודה אותה אתה מבצע. אם נמצאה התוכנה מתאימה לצרכיך, עליך לשלם למפתחיה תשלום סמלי (לפי הרשום בקבצי העזרה של כל תוכנה ותוכנה בנפרד) כדי לקבל רישיון מלא לשימוש בה. קבלת רישיון לשימוש בתוכנה יפתח בפניך מיגוון אפשרויות שלא עמדו לרשותך בהפעלת גרסת ה-ShareWare.

התקנת תוכנת גלישה לאינטרנט Microsoft Internet Explorer 5

תוכנית ההתקנה מזהה את גרסת מערכת ההפעלה ומתקינה את גרסת הדפדפן הדרושה. מומלץ להסיר גירסה קודמת של Internet Explorer, אם קיימת.

1. הכנס את התקליטור לכונן.
2. לחץ על לחצן **התחל** ובחר באפשרות **הפעלה**.
3. לחץ על לחצן **עיון**.
4. בחר בכונן התקליטורים בתיקיה Software\IE5 ובקובץ בשם SETUP.EXE.
5. לחץ על לחצן **פתח**. לחץ על לחצן **אישור**.
6. פעל לפי ההוראות על המסך.



אזהרה:

לפני ביצוע שדרוג מ-Windows 95 ל- Windows 98 בעברית (זו בה התפריטים בעברית ולחצן התחל מימין שורת המשימות), יש להסיר את Internet Explorer 5. לאחר השדרוג ניתן לבצע התקנה מחדש של הגירסה המתאימה.

FontsPekan

קובץ זה יתקין במחשב 2 גופנים בעברית לשימושכם. בסיום ההתקנה יש לבצע את הפעולות הבאות:

1. לחץ על **התחל**, הצבע על **הגדרות**, ובחר ב**לוח הבקרה**.
2. לחץ לחיצה כפולה על הסמל **גופנים**.
3. פתח את תפריט **קובץ** ובחר באפשרות **התקנת גופן חדש**.
4. עבור לתיקיה **C:\FontsPekan**.
5. לחץ על לחצן **בחר הכל** (סה"כ יש בתיקיה 2 גופנים).
6. ודא שתיבת הסימון **העתק גופנים לתיקיית הגופנים** מסומנת.
7. לחץ **אישור**.
8. סגור את חלון התיקיה **Fonts**.
9. סגור את חלון **לוח הבקרה**.

כעת, מוכנים הגופנים לשימוש בכל התוכנות המותקנות במחשב שלך: Word, Excel, PowerPoint וגם בתוכנות גרפיות, כגון Paint Shop Pro ו-PhotoShop.

הגופנים נקראים Tml-JUMP ו-Tml-step ויופיעו בתחתית רשימת שמות הגופנים (בדרך כלל). הרי דוגמה שלהם:

Tml-step

אבגדהוזחטיכךלמםנןסעפףצץקרשת1234567890

Tml-JUMP

אבגדהוזחטיכךלמםנןסעפףצץקרשת1234567890

NETEX

במקום לרשום **http://www.hod-ami.co.il** פשוט רישמו **הוד-עמי** והנה אתם באתר ההוצאה.

במטרה להגיע לאתר מסוים באינטרנט, שכתובתו אינה ידועה, אנו משתמשים בדרך-כלל באחת משתי דרכים: ניחוש של כתובת האתר ו/או פנייה לאינדקס או למנוע חיפוש

שתי הפעולות הן מסורבלות וגוזלות זמן ואנרגיה מיותרים. ניחוש הכתובת מחייב הקלדה של הכתובת המלאה באנגלית בדיוק מושלם, והוא עשוי להיות הליך גוזל זמן, במיוחד כאשר לא מצליחים למצוא את האתר בניסיון ראשון או בכלל.

עם netex לא צריך לנחש כתובות או לגלוש למנועי חיפוש כדי להגיע לאתר מסוים ברשת! פשוט מקלידים את שם האתר בעברית בחלון הכתובת בדפדפן, ומגיעים אליו ישירות.

אפשרויות השימוש במערכת

גלישה ישירה לאתר על-פי שמו או על-פי נתונים הקשורים בו.

מקלידים בחלון הכתובת של הדפדפן שם של אתר או חברה או מילות מפתח הקשורות באתר (בכל סדר שהוא), ומגיעים אליו ישירות.

לדוגמה:

- מקלידים הוד-עמי או הוצאת הוד-עמי או ספרי מחשבים ומגיעים ישירות לאתר הוצאת הוד-עמי לספרי מחשבים.
- מקלידים **בנק דיסקונט** - ומגיעים ישירות לאתר של בנק דיסקונט.
- מקלידים **סלקום** או **052** - ומגיעים ישירות לאתר של חברת סלקום;
- מקלידים **עכבר העיר** - ומגיעים ישירות לאתר של העכבר;
- מקלידים **144** - ומגיעים למודיעין 144 של בזק;

התחברות למערכת הניתוב החדשה של ישראל

כל שעליכם לעשות כדי להתחבר ל-NETEX הוא להתקין תוכנה קלה וחכמה:

1. יש לסגור את כל התוכנות הפתוחות כולל הדפדפנים הפועלים.
2. מתוך סייר Windows הפעילו את הקובץ **netex100.exe** שבתיקיה X:\Software\NetEx.
3. פעלו בהתאם להוראות.

בזמן ההתקנה התוכנה מזהה את הדפדפן/ים שמותקנים במחשב, והיא תפעל עם כולם. מייד אחרי סיום ההתקנה תוכלו להתחיל לגלוש חכם ובעברית.

המערכת שקופה למשתמש, כלומר היא "מתלבשת" על הדפדפן הרגיל ואינה נראית כלל. אין צורך להפעיל אותה או לבצע פעולה כלשהי כדי להשתמש בה: פשוט מקלידים את שם האתר המבוקש בשדה הכתובת של הדפדפן, ומגיעים ישר אליו.

התוכנה אינה מפריעה לעבודה רגילה עם הדפדפן. היא נכנסת לפעולה רק כאשר מקלידים נתון שאינו כתובת אינטרנט רגילה (URL). כאשר תקלידו **www.hod-ami.co.il** תגלשו ישירות לאתר **הוצאת הוד-עמי**, בדיוק כפי שנהגתם לגלוש לפני התקנת התוכנה (ללא מעורבות המערכת). אך אם תרצו, תוכלו להקליד **הוד-עמי** בשדה כתובת ולהגיע במהירות לאותו אתר בדיוק.

תיקיה ראשית SoftWare
(רשימה חלקית ועשויה להשתנות)

הערה: תוכנות להן יש גירסה מיוחדת עבור Windows 2000 יסומנו בתיקיה ששמה מסתיים ב- 2k (למשל, בתיקיה ICQ נמצא קובץ התקנה לתוכנה זו המתאים לכל גרסאות Windows, ובתיקיה ICQ2k נמצא קובץ התקנה עבור מערכת ההפעלה Windows 2000 בלבד).

בדרך כלל לחיצה כפולה על שם הקובץ המפורט ברשימה מפעילה את תוכנית ההתקנה.

שם תוכנה	תיאור
Adobe Acrobat	תוכנה לצפייה בקבצי pdf
Clean System	מחיקת קבצי dll שאין צורך בהם
FontsPekan	גופנים בעברית
ICQ	תוכנה לתקשורת אישית באינטרנט
MIRC	תוכנת הצ'אט הפופולרית ביותר ברשת
NetEx	תוכנה המאפשרת גלישה בעברית
Paint Shop Pro 6	תוכנה ליצירת, עיצוב ועיבוד תמונות
Power Toys	תוכניות שירות עבור Windows 9x
WinAmp	תוכנה להשמעת קבצי MP3 (מוסיקה)
WinZip	תוכנית לפריסה/דחיסה של קבצים
WordView	תוכנית לצפייה בקבצי doc

קרא את קובץ ONCD.DOC בתקליטור כדי לקבל מידע מעודכן על תכולת התקליטור המצורף.

קרא בנספח ו' כיצד להעתיק את קבצי המקור הרלוונטים לספר זה.

אינדקס

ב

ג

ט

י

כ

נ

ס

ע

צ

ק

ר

ש

ת